UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

APRENDIZAJE POR REFUERZO APROXIMADO EN EL VIDEOJUEGO SUPER MARIO BROS

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Javier Vecino Martínez

Tutora: Raquel Fuentetaja Pizán

Índice

Contenido

Índice de Ilustraciones

Índice de Tablas

1 Introducción

En los últimos años la industria de los videojuegos vive uno de sus momentos más dulces, llegando a superar en ingresos a industrias del entretenimiento tan consagradas como pueden ser el cine o la televisión. Esto se debe principalmente al imparable crecimiento tecnológico en los sistemas informáticos, especialmente en los procesadores gráficos, lo que ha permitido a los desarrolladores de videojuegos generar imágenes superrealistas que, en combinación con argumentos cada vez más elaborados, y una mayor capacidad para captar y expresar la acción dentro del juego, ofrece al usuario una experiencia muy cercana a vivir su propia "película" a través de su consola o PC.

A pesar de todas estas innovaciones, esta nueva generación de videojuegos sigue teniendo varios puntos en común con los videojuegos clásicos. Es el caso de los personajes controlados por la máquina o NPC (Non-Player Character), los cuales llevan décadas tratando de impedir que los jugadores consigan su meta en el juego, o por el contrario, tratando de ayudarles en sus aventuras. Dentro de la perspectiva de superrealismo anteriormente mencionada, y después de tantos títulos publicados a lo largo del tiempo, los jugadores exigen cada vez una mayor complejidad en la manera de interactuar con los NPC, los cuales deben ofrecer nuevas mecánicas y retos, así como movimientos cada vez más realistas, capaces de alimentar sus expectativas. Es en este punto donde intervienen los algoritmos de Inteligencia Artificial, y este a su vez, es uno de los factores que motivan la realización de este proyecto.

Este trabajo se basa en el simulador MarioAI, inspirado en el clásico juego Super Mario Bros, para el cual se ha desarrollado un agente automático capaz de jugar de forma similar a un humano y superar niveles por sí mismo buscando obtener la mayor puntuación posible. A lo largo de este documento se explica el planteamiento y realización de este agente, haciendo especial hincapié en el funcionamiento de los distintos algoritmos de Inteligencia Artificial usados para su entrenamiento, lo cual permite exponer a su vez un análisis de las ventajas y desventajas que ofrece cada uno de ellos.

2 Motivación y Objetivos

Como se puede deducir a partir de la introducción de este documento, el principal objetivo que persigue este proyecto consiste en la generación y entrenamiento de un agente automático capaz afrontar los niveles con una lógica similar a la de un humano, que busque obtener la mejor puntuación posible en cada uno de estos. Para ello se parte de un proyecto anterior realizado durante la asignatura Aprendizaje Automático impartida en el Grado de Ingeniería Informática, en el cual se desarrollaba un agente de Super Mario Bros con la técnica de aprendizaje por refuerzo Q-Learning. Siguiendo esta línea base de trabajo, este proyecto tiene los siguientes objetivos y motivaciones:

- Realizar un nuevo agente basándose en el anterior, aplicando esta vez la técnica de Q-Learning Aproximado y tratando de lograr el mejor comportamiento y rendimiento posible sobre los distintos niveles del juego.

- Profundizar aún más en el algoritmo de Q-Learning, así como en el uso de la técnica de clustering.

- Estudiar y aplicar la técnica de Q-Learning Aproximado como alternativa al clustering sobre el problema propuesto.

- Comparar los resultados obtenidos por ambas técnicas y observar en qué aspectos del problema se adapta mejor cada una de ellas.

- Explorar distintas formas de abordar un mismo problema, realizando una experimentación sobre la generación y tratamiento de los datos de entrada del problema.

El estudio y uso de estas técnicas de cara al desarrollo de videojuegos resulta muy interesante desde el punto de vista de la Inteligencia Artificial, dado que permiten desarrollar movimientos y comportamientos más realistas en los personajes controlados por la máquina, haciéndolos menos predecibles por el jugador. Pero estas no son sólo exclusivas para el desarrollo de videojuegos, y su análisis resulta también muy interesante de cara a ser extrapoladas a una buena cantidad de áreas mediante, por ejemplo, la robótica, la cual se está extendiendo poco a poco en sectores como la hostelería, la medicina o el ocio. Aplicadas a este campo podrían manejar de forma automática las acciones y el comportamiento de un robot frente a las distintas situaciones a las que se pueda enfrentar en su entorno.

3 Estado de la cuestión

En este apartado se abordan los aspectos teóricos de las técnicas que se han empleado, se describe el software utilizado como base para el problema y se introduce el juego Super Mario Bros y sus aspectos más relevantes de cara a la resolución del problema.

3.1 El juego: Super Mario Bros

Super Mario Bros es un juego de plataformas y aventuras diseñado por Shigeru Miyamoto y lanzado en 1985 por la compañía Nintendo para su consola NES. Tiene sus orígenes en el personaje "Jump Man" que aparecía en el juego Donkey Kong (1981), el cual tenía que salvar a la princesa Vilma de las garras del gorila. Nintendo decidió desarrollar este personaje dándole una identidad propia, Mario, y creando a su hermano Luigi para su siguiente título, Mario Bros (1982). En este juego, desarrollado para máquinas arcade, los dos hermanos tenían que mantener limpias las tuberías de un mundo subterráneo, eliminando para ello a los enemigos que iban saliendo de ellas. Tuvo un éxito modesto, pero se siguió desarrollando la idea para la siguiente entrega, Super Mario Bros, en la cual se incluyó el Reino Champiñón, dotando al juego de un montón de niveles nuevos repletos además de secretos, y mejorando su narrativa, lo que lo hacía único con respecto a sus competidores. Rápidamente se convirtió en un éxito de ventas y cosechó una enorme popularidad, lo que provocó que se lanzasen varias secuelas del mismo, así como versiones inspiradas en la saga y personajes de Mario Bros para las distintas consolas que Nintendo ha desarrollado a lo largo del tiempo, convirtiendo al célebre fontanero en el icono por excelencia de la compañía y probablemente, en la cara más conocida de los videojuegos. A día de hoy aún se siguen lanzando nuevas entregas inspiradas en el juego original, haciendo de este uno de los más vendidos de la historia [1] [2].

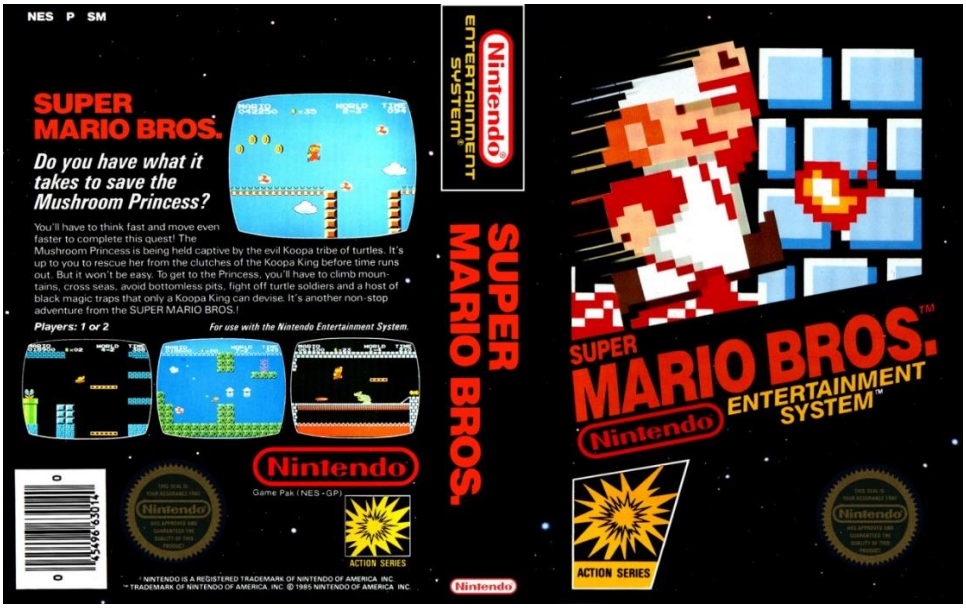

Ilustración 1: Portada y contraportada de la caja de Super Mario Bros

Este proyecto se centra en superar las mecánicas de juego proporcionadas por el simulador MarioAI (ver apartado El simulador: MarioAI (Infinite Mario Bros)), las cuales se corresponden casi en su totalidad con las del Super Mario Bros original. En él, el objetivo principal de Mario es rescatar a la princesa Peach, la cual se encuentra al final de cada nivel en el simulador (en el original se encuentra sólo en el último nivel del último mundo), para lo que tendrá que sortear una serie de obstáculos y eliminar a aquellos enemigos que le impidan llegar a su meta antes de que se acabe el tiempo. A su vez, por el camino encontrará monedas que le permitirán obtener más puntuación (en el original además 100 monedas otorgan una vida extra) y bloques con interrogante que al ser golpeados le pueden proporcionar monedas extra o poderes adicionales:

- Si Mario se encuentra en su tamaño original el bloque puede liberar una seta que al ser atravesada le hace más grande, le permite romper bloques de ladrillos y además le hace resistente a un golpe adicional de los enemigos.

- Si Mario atravesó una seta anteriormente, el bloque puede liberar una flor de fuego que al ser atravesada hace que entre en Modo fuego otorgándole la capacidad de disparar bolas de fuego y haciéndole resistente a dos golpes de los enemigos.

Los enemigos que Mario puede encontrar en el simulador a lo largo de los distintos niveles son los siguientes:

- **Goomba**: Es el enemigo más común del juego. Los Goombas son unos champiñones con pies que se mueven de lado a lado en algunas zonas del nivel y dañan a Mario al entrar en contacto con él. Para eliminarlos se puede saltar encima de ellos o dispararles, en caso de que Mario se encuentre en Modo Fuego.

Ilustración 2: Artwork e imagen in-game de un Goomba

- **Koopa Troopa**: Los Koopa Troopas son unas tortugas que, al igual que los Goombas, se mueven de lado a lado en algunas zonas del nivel y dañan a Mario al entrar en contacto con él. A diferencia de estos, al saltar sobre los Koopa Troopas no se les mata si no que se les mete dentro del caparazón, el cual puede ser disparado contra otros enemigos tras saltar una vez más sobre él. Sólo se puede matar a un Koopa Troopa en el Modo Fuego mediante un disparo a él o al caparazón, en caso de haber saltado sobre este previamente, si no, si el caparazón no está en movimiento volverá a salir al cabo de un tiempo. En algunos niveles estos enemigos tienen la capacidad de volar y hará falta un salto adicional para derribarles, o en su defecto, dispararles directamente.

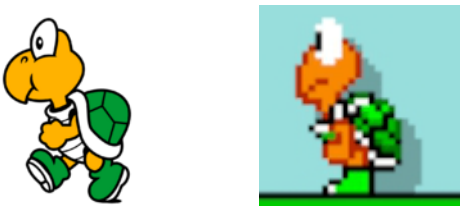

Ilustración 3: Artwork e imagen in-game de un Koopa Troopa

- **Bill Bala**: Los Bill Bala son balas disparadas en línea recta desde los cañones Lanza Bill. Son inmunes a las bolas de fuego y solo se pueden derrotar saltando sobre ellos o lanzándoles un caparazón de Koopa.

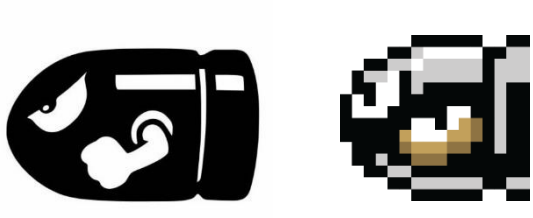

Ilustración 4: Artwork e imagen in-game de Bill Bala

Por otra parte, algunos de los obstáculos que se puede encontrar pueden requerir de alguna estrategia adicional además de dar un salto para salvarlos:

- **Bloques**: Hay dos tipos de bloques: los bloques de ladrillos y los bloques con interrogante. Los bloques con interrogante, como ya se explicó anteriormente, otorgan un beneficio, pero son irrompibles independientemente del modo en el que se encuentre Mario. Los bloques de ladrillos, sin embargo, se pueden romper en Modo Grande (tras atravesar una seta) saltando debajo de ellos, lo que en ocasiones permite a Mario trazar una nueva ruta para sortear, por ejemplo, un grupo de enemigos.

Ilustración 5: Imagenes in-game de un bloque de ladrillos y un bloque con interrogante

- **Tuberías**: Por lo general las tuberías que se encuentran en los niveles se pueden sortear saltando y avanzando sobre ellas sin más (en el juego original algunas también permiten bajar por ellas y descubrir atajos o niveles ocultos, pero en el simulador esto no ocurre). Sin embargo, en algunos niveles las tuberías ocultan una planta carnívora que sale de la misma y ataca a Mario si se encuentra sobre ella o si cae sobre la planta al saltar. Esta planta puede ser derrotada solo mediante un disparo en Modo Fuego, de lo contrario solo puede ser esquivada mediante un salto.

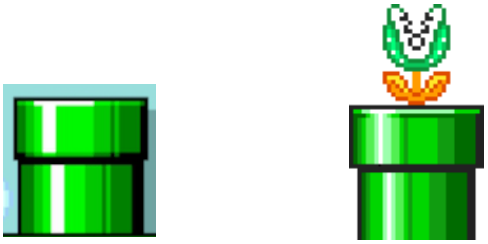

Ilustración 6: Imagenes in-game de una tuberia normal y otra con planta carnívora

- **Lanza Bill**: Los Lanza Bill son los cañones que lanzan enemigos Bill Bala cada cierto intervalo de tiempo, y suelen ser dobles, teniendo un cañón a la izquierda y otro a la derecha. Pueden ser sorteados mediante un salto mientras no se produzcan disparos.

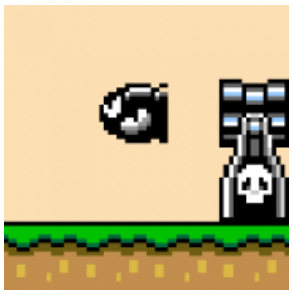

Ilustración 7: Lanza Bill disparando un Bill Bala

3.2 El simulador: MarioAI (Infinite Mario Bros)

Para la realización de este proyecto se ha utilizado un simulador de Super Mario Bros llamado MarioAI. Este simulador desarrollado por Sergey Karakovskiy y Julian Togelius [3] está basado en la versión en Java de Super Mario desarrollada por Markus Persson, llamada Infinite Mario Bros, incluyendo algunas modificaciones para ser empleado en competiciones de Inteligencia Artificial.

Este simulador presenta tres niveles de dificultad (0-mínima, 1-media, 2-alta) y 2^{31}-1 semillas de generación de escenarios por cada uno de ellos. A su vez, en cada nivel de dificultad se van incluyendo distintos tipos de enemigos y obstáculos más difíciles de eliminar y superar. Todo esto se traduce en una gran cantidad de escenarios para poner a prueba los agentes generados, así como para la recogida de datos para desarrollarlos. En cada instante (*tick*) de la partida que se juega, el simulador proporciona tres matrices de datos de 19x19 casillas que describen el entorno cercano a Mario, el cual se encuentra en la casilla central (casilla 9, 9) dentro de éstas, quedando el resto de casillas dedicadas a situar los diferentes elementos que le rodean según la matriz. Dichas matrices y sus peculiaridades son:

- **Matriz de escena (Level Scene Observation)**: En ella se sitúan solo los elementos del escenario como tuberías, cañones, bloques o laderas.

- **Matriz de enemigos (Enemies Observation)**: En ella se sitúan solo los distintos enemigos, las bolas de fuego, las flores y los champiñones.

- **Matriz fusionada (Merged Observation)**: Es una combinación entre ambas matrices, en ella se muestran tanto enemigos como elementos del escenario al mismo tiempo.

A su vez, estas matrices disponen de tres niveles (0, 1, 2) de generalización, es decir, pueden ser más específicas describiendo cada elemento o generalizarlos en grupos más grandes:

- **Nivel 0 (Generalización baja)**: El nivel de generalización es mínimo distinguiendo a casi todos los tipos de enemigos y obstáculos asignándoles un código único a cada uno. Los aspectos de clasificación más relevantes de este nivel son:

  - Distinción entre bloques: se pueden romper o no.

  - Separa los cañones en dos partes: la boca y la estructura.

  - Clasifica las laderas dependiendo de si se pueden saltar o no.

  - Separa las escaleras en dos partes: la parte superior y la base.

  - Cada enemigo tiene un código asignado, al igual que el resto de elementos del escenario no mencionados anteriormente.

- **Nivel 1 (Generalización intermedia)**: Se codifican enemigos y obstáculos en grupos más relevantes:

  - Se generalizan todos los tipos de bloque bajo un tipo, independientemente de si se pueden romper o no.

  - Se clasifican las laderas dependiendo de si se pueden saltar o no.

  - Clasifica en un mismo tipo a aquellos obstáculos que pueden dañar a Mario, como las flores o los cañones.

  - Clasifica los enemigos dependiendo de si se les puede matar saltando sobre ellos o no.

  - Se separan las escaleras en dos partes: parte superior y base.

  - El resto de elemento de la escena que no hayan sido mencionados reciben un código único.

- **Nivel 2 (Generalización alta)**: Se reportan cada uno de los siguientes grupos:

  - Todos los tipos de enemigos bajo el mismo código.

  - Existencia de monedas.

  - Todos los tipos de obstáculos bajo un mismo código.

Un ejemplo de la codificación que reciben los distintos elementos del simulador es la siguiente según la codificación del nivel 1:

- -24: Bloque.
- -60: Ladera.
- -62: Ladera no sorteable.
- -85: Tubería.
- 0: Vacío o irrelevante.
- 2: Moneda o seta.
- 3: Flor de fuego.
- 25: Bola de fuego lanzada por Mario.
- 80: Enemigo que se puede matar de un salto.
- 93: Enemigo que no se puede eliminar mediante salto.

Estas matrices y sus distintos modos de generalización permiten realizar un tratamiento de datos para describir posteriormente estados o situaciones en las que se encuentra Mario a lo largo de cada partida. En la siguiente ilustración (Ver Ilustración 8) se puede observar una representación de una de las matrices descritas anteriormente, durante una partida.

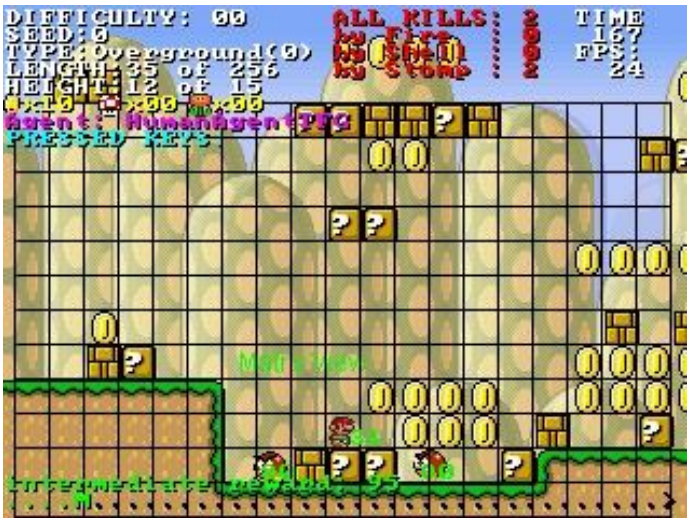

Ilustración 8: Representación gráfica de la interfaz, incluyendo la matriz de datos, en la que se pueden observar los códigos que reciben algunos de los elementos.

3.2.1 Estado del arte: MarioAI Championship.

Desde 2009 hasta 2012 se han realizado competiciones anuales de jugadores automáticos utilizando el simulador MarioAI organizadas por los desarrolladores del mismo. Las diferentes fases de estas han tenido lugar en diferentes congresos internacionales sobre Inteligencia Artificial e Inteligencia Computacional. La primera competición, en 2009, consistía en generar agentes automáticos que fuesen capaces de superar el mayor número de niveles posibles. A partir de la siguiente competición, en 2010, se crearon más categorías para poner a prueba más tipos de agentes. Son las siguientes [4]:

- **Gameplay**: Esta categoría persigue el mismo objetivo que la competición original de 2009: en ella se encontrarán aquellos agentes diseñados para pasarse el mayor número de niveles posibles.

- **Learning**: En esta categoría se evalúan aquellos agentes basados en aprendizaje. Para llevar a cabo la evaluación se pone a prueba la capacidad de aprender de estos enfrentándolos a niveles que no hayan visto durante la fase de entrenamiento. Las reglas especifican que cada agente puede entrenar cada nivel 10000 veces, contando solo para la competición la iteración 10001. De esta forma, se premia más a aquellos agentes con más habilidad para aprender a especializarse.

- **Turing test**: Esta categoría comprende aquellos agentes diseñados para imitar la forma de jugar de los seres humanos. En la prueba de evaluación se presenta ante una audiencia no experta una serie de videos de humanos y agentes automáticos jugando simultáneamente un mismo nivel. La audiencia debe votar por cada vídeo cuál jugador creen que es humano y cuál creen que es una máquina.

- **Level Generation**: Esta categoría se centra en aquellos competidores que han empleado el simulador MarioAI para generar nuevos niveles a partir de la información recibida sobre el estilo y las capacidades de juego de un humano. Para probar estos generadores, se invita a un humano a jugar un nivel de prueba y a continuación los generadores, en turnos de dos, tienen que producir un nivel que se ajuste al comportamiento de juego de la persona. El jugador decidirá tras jugar los niveles generados, cual le ha resultado más atractivo.

De las diferentes categorías de esta competición han salido agentes con muy buenos resultados que utilizan y adaptan algoritmos muy distintos para la resolución del mismo problema. Algunos de los más interesantes son los siguientes:

- **Robin Baumgarten** ganó la competición del año 2009 (categoría Gameplay) con un agente que utiliza el algoritmo de búsqueda A*. Para aplicar esta técnica trató el problema como si fuese uno de búsqueda de caminos, de tal forma que, mediante un simulador de físicas, se trazan aquellas rutas seguras que puede seguir Mario a partir de su posición y el algoritmo se encarga de seleccionar la que le lleve más

lejos. La siguiente ilustración (ver Ilustración 9) muestra cómo traza las rutas este agente:

Ilustración 9: Posibles rutas trazadas por el agente A de Robin Baumgarten*

- El ganador de la competición del año 2010 fue el agente REALM, presentado en la categoría Gameplay y desarrollado por **Slawomir Bojarski** y **Clare Bates Congdon**. Este jugador automático está basado en el uso de un conjunto de 20 reglas que buscan maximizar la distancia que recorre Mario en cada nivel estableciendo precondiciones para las distintas situaciones en las que se puede encontrar a lo largo de cada uno.

- **Erek Speed**, **Stephie Wu** y **Tom Lieber** presentaron un agente en la categoría Learning, donde la política a llevar a cabo en cada situación se optimizaba mediante el algoritmo de búsqueda estocástica "Busqueda del Cuco". Partiendo de las políticas establecidas en el agente preprogramado ForwardJumpingAgent, en el cual solo se avanza y se salta hacia delante, el algoritmo de búsqueda identifica en que situaciones se debería modificar la política de acción y procede a modificarla, mediante mutaciones que eligen acciones aleatorias, en cada una de estas. De esta forma, el agente es capaz de encontrar bloques ocultos y evitar quedarse bloqueado por obstáculos.

- **Laura Villalobos** por su parte presentó un agente basado en programación genética, dividiendo las 10000 iteraciones de la prueba en 25 generaciones de 400 individuos, usando una representación basada en árboles y un conjunto de instrucciones de programación genética. Las entradas al problema son los elementos que componen la matriz de observación del simulador.

En este proyecto se presenta un agente desarrollado mediante la técnica de aprendizaje por refuerzo Q-Learning Aproximado, el cual pretende aportar una solución óptima del problema diferente a la de la combinación de Q-Learning y

Clustering, utilizada por el agente realizado en la asignatura Aprendizaje Automático. Todas estas técnicas se detallarán en los siguientes apartados de este documento.

3.3 Software base: Q-Learning

Además del simulador MarioAI, para la realización de este proyecto se ha usado como punto de partida el software Q-Learning, desarrollado por el grupo PLG de la Universidad Carlos III de Madrid en lenguaje Java. Este programa contiene todas las funciones asociadas al algoritmo de Q-Learning (ver apartado Q-Learning) y se usó en la práctica de la que parte este proyecto para aplicar dicha técnica a un agente de Super Mario. En el desarrollo de este trabajo se han realizado modificaciones sobre este software, añadiendo funciones adicionales para que también soporte el uso de la técnica Q-Learning Aproximado (ver apartado Q-Learning Aproximado.), añadiendo las funciones pertinentes.

3.4 Aprendizaje Automático

El Aprendizaje Automático [5] es una rama de la Inteligencia Artificial que se especializa en el desarrollo y uso de técnicas para darle a las máquinas la capacidad de *aprender*. Esto se realiza mediante la creación de programas capaces de generalizar y reconocer comportamientos a partir de ejemplos obtenidos del análisis de datos (proceso de inducción), así como la extracción de conocimiento sobre propiedades no observadas en un objeto o individuo a partir de aquellas que sí han sido observadas previamente (predicción). Para que estos algoritmos sean capaces de realizar estas tareas se les dota con la capacidad de reconocer patrones a partir de ejemplos, enseñándoles para ello los pasos de aprendizaje necesarios (de manera similar a la que un experto humano describe su modo de realizar una tarea), de forma que tras un entrenamiento sea capaz de aplicar los patrones aprendidos y realizar la tarea ante distintas situaciones o casos. De esta manera se posibilita que la máquina aprenda a partir de la experiencia con el entorno en vez de simplemente reconocer patrones programados a priori. Estos algoritmos se clasifican en los siguientes grupos:

- **Aprendizaje supervisado**: Estos algoritmos deducen una función de correspondencia entre entradas y salidas a partir de un conjunto de datos de entrenamiento. Típicamente son usados para problemas de clasificación o regresión.

- **Aprendizaje no supervisado**: Los algoritmos de aprendizaje no supervisado difieren de los supervisados al no tener conocimiento a priori sobre las etiquetas, por lo que en los ejemplos de entrada no aparecen las salidas que estos producen, teniendo que ser capaz el algoritmo de agrupar ejemplos similares. A su vez estos algoritmos son capaces de auto-organizarse sin la necesidad de que exista la figura de un “profesor”. Son típicamente utilizados en problemas de agrupación y compresión de datos.

- **Aprendizaje semisupervisado**: Se trata de una combinación entre algoritmos supervisados y no supervisados para poder realizar tareas de clasificación cuando parte de los ejemplos de entrenamiento no están etiquetados.

- **Aprendizaje por refuerzo**: Son algoritmos basados en el feedback que obtienen de un entorno al realizar acciones. Este feedback puede resultar ser positivo o negativo, por tanto, se trata de algoritmos basados en el aprendizaje a base de ensayo-error. Son útiles en problemas en los que se requiera aprender una política de movimientos para cada posible situación.

De todas las técnicas de aprendizaje automático descritas anteriormente, este proyecto se centra en la aplicación del aprendizaje por refuerzo, por tanto, en los siguientes apartados se profundizará en los conceptos que maneja así como en la definición del mismo.

3.5 Aprendizaje por refuerzo

Como se mencionó en el anterior apartado, el aprendizaje por refuerzo se centra en el tratamiento del feedback que reporta realizar acciones, de forma que si este es positivo, se premie al agente favoreciendo el uso de una acción en concreto para un determinado caso, o por el contrario penalizándolo para evitar su uso. Esto se formula como un proceso de decisión de Markov (MDP) con el siguiente modelo [6] [7]:

- Un **espacio de estados** S, donde cada estado representaría una situación determinada del problema. Estos estados se generan a partir de los valores que toman ciertos atributos o características del problema.

- Un **espacio de acciones** A, en el que están contenidas todas las acciones que se pueden realizar en el problema. La ejecución de estas acciones da lugar a la transición entre los distintos estados.
- Una **función de transición** entre estados $T: S\ x\ A \rightarrow P(S)$, donde $P(S)$ es la probabilidad de transitar a un estado perteneciente espacio de estados S. En el caso general esta función es estocástica, por lo que la ejecución del mismo par estado/acción tiene una probabilidad de producir distintos resultados, con lo cual $T(s, a, s')$ describe la probabilidad de transitar desde el estado s al estado s' ejecutando la acción a. En el caso de que se trate de un MDP determinista la ejecución cada par estado/acción siempre obtendrá la misma transición.

- Una **función de refuerzo** R, que mide el feedback de la acción realizada a partir de un estado y premia o penaliza al problema en función de si es positiva o negativa. Esta función es estocástica en el caso general, la ejecución de un par estado/acción puede producir distinto refuerzo, mientras que en el caso de un MDP determinista siempre produce el mismo refuerzo para cada par estado/acción.

En las siguientes ilustraciones (Ilustración 10 e Ilustración 11) se puede observar una representación gráfica de cada tipo de MDP:

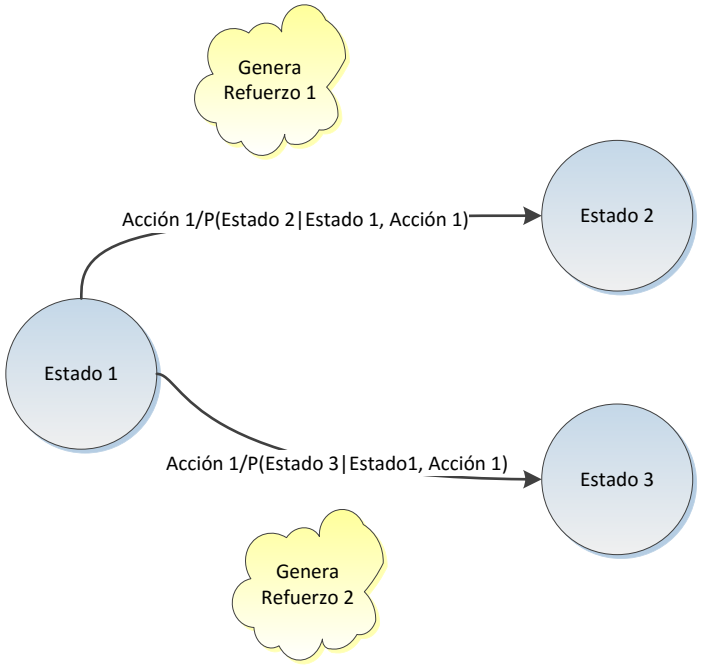

Ilustración 10: Ejemplo de un MDP estocástico.

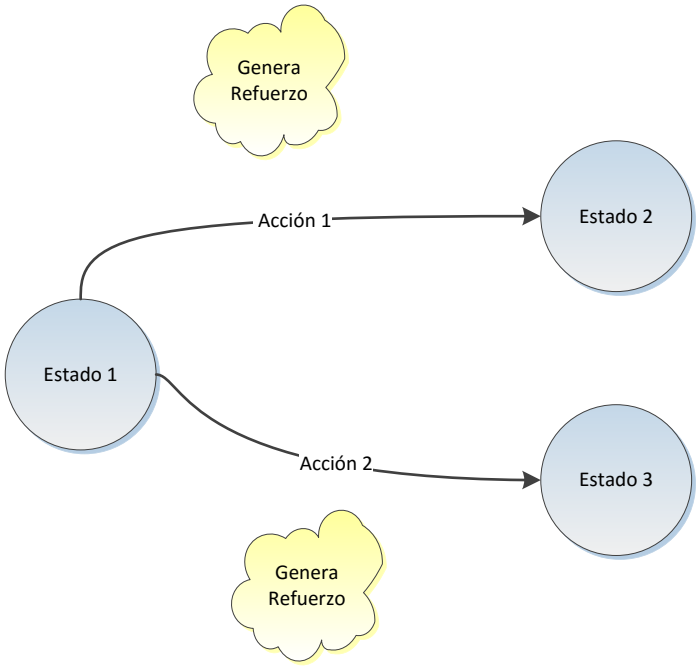

Ilustración 11: Ejemplo de un MDP determinista.

El aprendizaje por refuerzo se aplica cuando la función de transición entre estados es desconocida y no es sencillo aprenderla. Se basa en generar la política de movimientos a partir de experiencias que se obtienen explorando el entorno. Estas experiencias se recogen en unas tuplas de aprendizaje con el siguiente aspecto: (s, a, s', r), donde s es el estado inicial en la tupla, a es la acción ejecutada, s' el estado al que transita tras ejecutar la acción y r el refuerzo generado. Para explorar el entorno del problema y generar estas tuplas se definen unas estrategias de explotación y exploración sobre este, las cuales son las encargadas de determinar las acciones que realiza el agente en los ejemplos de entrenamiento. Es necesario escoger una estrategia adecuada al problema, dado que no todas escalan igual de bien con el tamaño del espacio de estados, y una mala colección de tuplas puede dar lugar a resultados con muy bajo rendimiento. Una de las estrategias más usadas es ϵ-greedy, la cual es un método de exploración simple que alterna entre escoger acciones aleatorias y acciones que supone tienen un mejor efecto sobre el problema, según una probabilidad asignada a cada una de estas. La Ilustración 12 representa gráficamente el proceso de exploración del entorno.

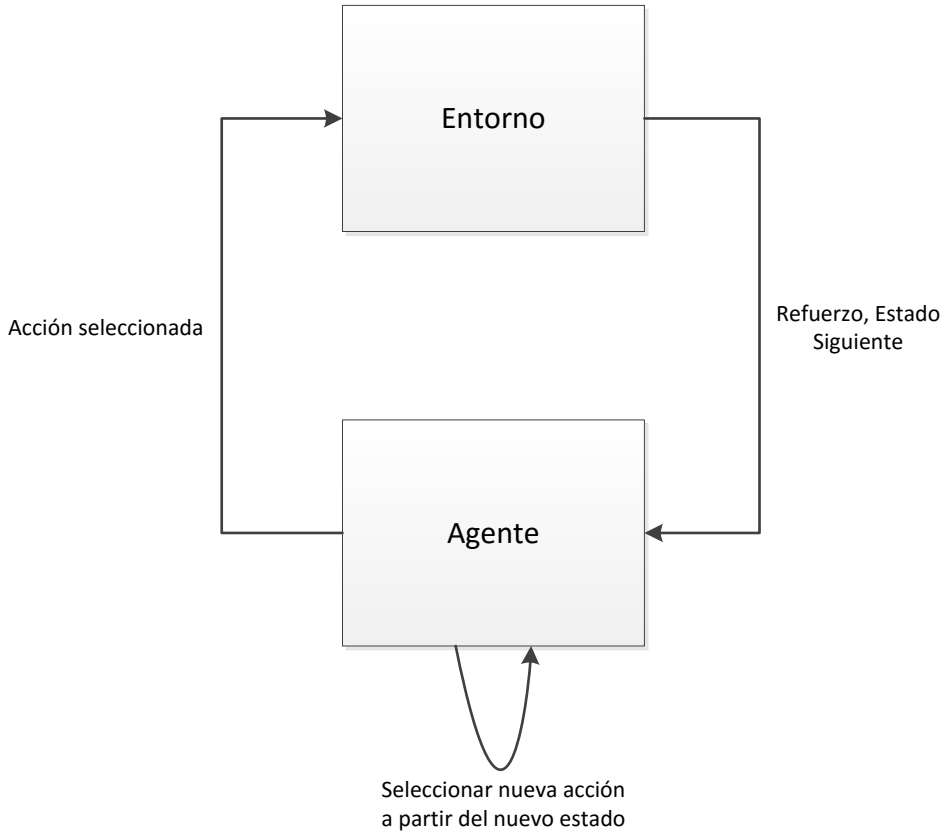

Ilustración 12: Representación del proceso de exploración del entorno

Una vez creado todo el conjunto de tuplas faltaría procesarlas para obtener cual es la mejor acción para cada momento del problema. Esto se puede realizar de dos maneras:

- Mediante **aprendizaje off line** se llevaría a cabo una fase de entrenamiento previa a la ejecución del problema. En dicha fase, el algoritmo escogido visitaría todas las tuplas, generaría la política, y dejaría ya decidido cual acción corresponde a cada estado.

- Mediante **aprendizaje on line**, en cambio, recibiría información del estado en el que se encuentra el agente mientras se ejecuta el problema, actualizando su política dinámicamente. De este modo, la acción correspondiente a un estado no quedaría fijada en una fase de entrenamiento previa, si no que se decidiría durante la ejecución, quedando el par estado-acción sujeto a cambios según la información que se le va presentando al algoritmo.

Algunos de los algoritmos más populares en el aprendizaje por refuerzo son los siguientes:

- **Q-Learning**: Es un algoritmo de aproximación libre de modelo: no conoce las funciones de transición de estado ni de refuerzo. Dado un MDP finito, este método intenta aprender mediante ensayo y error una política de acciones óptima de manera *off-policy*, es decir, de manera independiente a las acciones ejecutadas por el agente [6] [8].

- **Dyna-Q**: Pese a ser un algoritmo muy similar a Q-Learning, Dyna-Q presenta algunas diferencias con respecto a este. Una de ellas es que es un algoritmo basado en el modelo, es decir, intenta aprender este en cada paso que da, utilizándolo a su vez para realizar actualizaciones sobre su función Q [6].

- **SARSA**: Es un algoritmo basado en el modelo que debe su nombre a la forma que tienen sus tuplas de aprendizaje: STATE-ACTION-REWARD-STATE'-ACTION'. Se caracteriza por ser un método *on-policy*, es decir, aprende la política de selección de acciones a partir de las acciones que va ejecutando el agente [8].

De todos estos algoritmos, este proyecto se va a centrar en uno en concreto: Q-Learning, y una variante del mismo: Q-Learning Aproximado, los cuales se explicarán en los siguientes apartados de este documento.

3.6 Q-Learning

Q-Learning es un método *off-policy* de aproximación libre de modelo que consiste en aprender a decidir ante una situación determinada cuál es la acción más adecuada para lograr un objetivo. Partiendo de un modelo MDP, Q-Learning genera una tabla llamada Tabla Q con tantas filas como estados y tantas columnas como acciones, que contiene en cada casilla el refuerzo esperado acumulado en el tiempo para cada par estado-acción. Esta tabla se consultará posteriormente durante la ejecución del problema para determinar cuál es la mejor acción que puede llevar a cabo el agente según el estado en el que se encuentre. Para generarla el algoritmo sigue los siguientes pasos: inicializa los valores de la tabla a un valor (generalmente a 0), después toma una

tupla de experiencia y actualiza la Tabla Q con la información contenida en ella, y repite estos dos últimos pasos hasta que los valores de la tabla convergen, obteniendo así la política de comportamiento óptima a llevar a cabo, teniendo en cuenta que el algoritmo busca la maximización del refuerzo esperado a largo plazo. La actualización de la Tabla Q se lleva a cabo siguiendo una función que varía en función de si el MDP es determinista (una única posibilidad de transitar tras ejecutar una acción) o no determinista (distintas transiciones al ejecutar una acción, basadas en probabilidad) [6] [9].

- **Función de actualización determinista**:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

  - s = Estado inicial.
  - a = Acción inicial.
  - r = Refuerzo.
  - γ = Tasa de descuento.
  - s' = Estado siguiente.
  - a' = Acción asociada al estado siguiente.

- **Función de actualización no determinista**:

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha\, [r + \gamma \max_{a'} Q(s',a')]$$

  - α = Tasa de aprendizaje.
  - s = Estado inicial.
  - a = Acción inicial.
  - r = Refuerzo.
  - γ = Tasa de descuento.
  - s' = Estado siguiente.
  - a' = Acción asociada al estado siguiente.

Como se puede observar ambas funciones se basan principalmente en actualizar el valor Q de cada par estado-acción con el refuerzo obtenido al ejecutar la acción y la recompensa proporcionada de la ejecución del estado siguiente con la acción que genera el máximo valor Q para este último. La única diferencia entre ambas funciones está en cómo tratan el valor Q almacenado en la tabla en la anterior iteración: la función determinista no necesita acumular el valor anterior dado que el refuerzo para un estado y una acción siempre será el mismo, pero en el caso de la función no determinista sí es necesario dado que la ejecución del mismo par estado-acción puede generar distintos valores en el refuerzo. También se puede observar que hay dos tasas que afectan a la manera en la que actualizan ambas funciones, la tarea asignada a cada una es la siguiente:

- **Tasa de aprendizaje (α)**: Esta variable determina cuanta información acumulada en anteriores iteraciones se conserva en cada actualización, lo que controla la velocidad de aprendizaje. Su valor oscila entre 0 y 1, siendo 1 el valor que haría que el agente sólo tuviera en cuenta la nueva información (óptimo en un entorno determinista), y 0 el valor que impediría aprender al agente.

- **Tasa de descuento (γ)**: El valor de esta variable influye en el peso que se le da en la actualización a las recompensas obtenidas del estado siguiente. Un valor de 0 haría que el agente descartase por completo toda recompensa futura y solo aprendiese del refuerzo actual. Los valores cercanos o mayores que 1 pueden provocar que los valores de las recompensas sean demasiado altos y diverjan.

Pese a que está demostrado que dado cualquier MDP finito el método Q-Learning es capaz de encontrar una política óptima, la viabilidad del algoritmo puede verse comprometida con espacios de estados o acciones muy grandes al tener que almacenar sus resultados en la Tabla Q, la cual podría alcanzar un tamaño desorbitado. En estos casos es necesario combinar este algoritmo con otras técnicas que simplifiquen dichos espacios o que mejoren su escalabilidad. Dos posibles soluciones a esta problemática son las siguientes:

- **Clustering**: El clustering consiste en agrupar elementos que tienen cierta similitud en conjuntos o clusters. En este caso, esta técnica se aplicaría sobre aquel espacio que esté causando problemas al algoritmo debido a su tamaño.

- **Q-Learning Aproximado**: Esta técnica combina una función de aproximación con el algoritmo Q-Learning haciéndolo capaz de enfrentarse a espacios de estados o acciones muy grandes.

Debido a las características que presenta el espacio de estados del problema abordado en este proyecto se han probado estos dos métodos, por lo que en los siguientes apartados se realizará un análisis más exhaustivo de ambas técnicas.

3.7 Clustering

Se llama Clustering a la tarea de etiquetar un conjunto de datos que guarden cierta similitud entre sí bajo una misma clase o grupo (cluster), distinguiéndolos así del resto de elementos del conjunto total, los cuales pueden pertenecer a su vez a otros clusters. Al tratarse de un proceso que no tiene conocimiento sobre alguna variable objetivo que pueda relacionar los datos entre sí, si no tiene que ser el propio algoritmo el que establezca una relación entre variables similares, se considera el Clustering una tarea de aprendizaje no-supervisado [10].

No existe una única regla o algoritmo para realizar este proceso, sino que es necesario experimentar con las capacidades de cada algoritmo, o incluso combinar varios de ellos, para encontrar una relación correcta de los datos. A su vez, estos algoritmos no

responden siempre al mismo criterio de agrupación, sino que existen un buen número de criterios, algunos de los cuales son los siguientes [11]:

- **Agrupamiento basado en conectividad (agrupamiento jerárquico):** La principal idea que manejan los algoritmos de agrupamiento jerárquico es que aquellos elementos que están más cerca unos de otros, guardan más relación que aquellos que están alejados. Estos algoritmos difieren entre sí en cómo están calculadas las distancias, con lo que dejan a cargo del usuario la elección de la función de cálculo de distancia, así como el criterio de conexión entre elementos. Dado que son muy poco robustos con el ruido y a que, por lo general, debido a su complejidad, suelen ser bastante lentos, están considerados obsoletos en minería de datos. En la Ilustración 13 se muestra una representación gráfica de esta técnica.

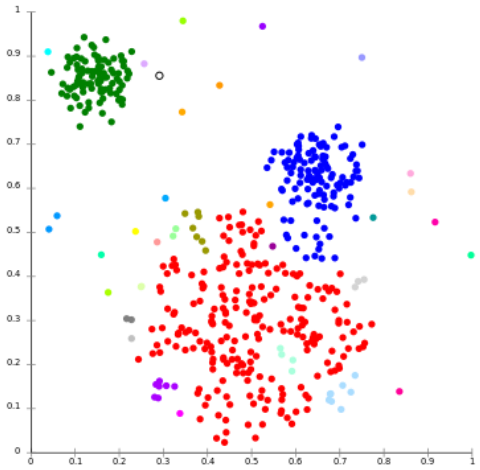

Ilustración 13: Datos gaussianos agrupados con criterio de Enlace Simple (uno de los más populares). Se han generado 35 grupos en este caso.

- **Agrupamiento basado en centroide**: Este tipo de agrupamiento se basa en que los grupos están representados por un vector central, el cual no tiene por qué pertenecer al conjunto de datos. Teniendo en cuenta esta base, el algoritmo K-Means, el principal de esta categoría, resuelve la agrupación como si se tratase de un problema de optimización en el cual hay que encontrar los centroides de los grupos (el número de grupos lo especifica el usuario previamente) para después asignar cada elemento al centroide más cercano. La Ilustración 14 muestra una representación gráfica de esta técnica.

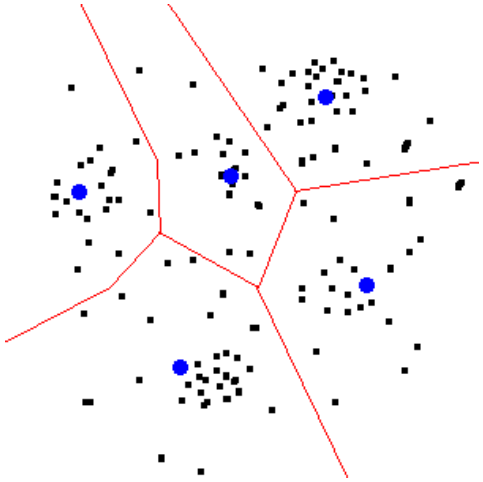

Ilustración 14: Distribución de los datos en clusters según K-Means. Los puntos azules denotan los centroides asociados a cada cluster.

- **Agrupamiento basado en distribuciones**: Este modelo de agrupamiento hace uso de la estadística y crea los clusters asignándole a cada objeto una probababilidad de pertenecer a una determinada distribución. Uno de los algoritmos más usados dentro de esta categoría es el EM (Expectation-Maximization), el cual hace uso de distribuciones Gaussianas para clasificar el conjunto de datos. La Ilustración 15 muestra una representación gráfica de esta técnica.

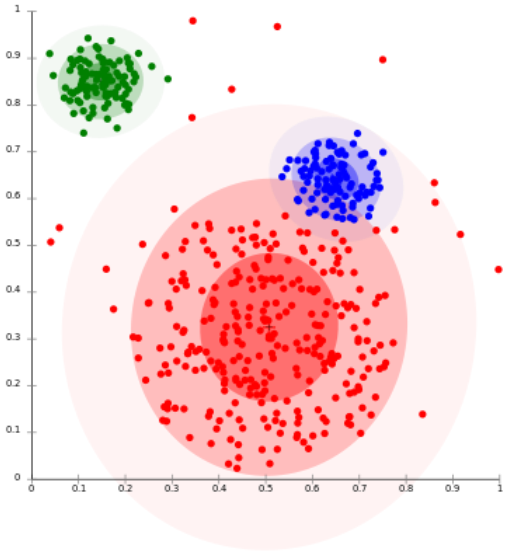

Ilustración 15: Clusters generados mediante distribuciones Gaussianas por el algoritmo Expectation-Maximization.

- **Agrupamiento basado en densidad**: El agrupamiento basado en densidad se basa en distinguir grupos con un área de densidad mayor a la del resto del conjunto de datos. El algoritmo más popular dentro de la categoría es DBSCAN, el cual se basa en conectar aquellos elementos dentro de cierto umbral de distancia que además cumplan el criterio de densidad escogido, por ejemplo, que haya un número

mínimo de elementos en un determinado radio. La Ilustración 16 muestra una representación gráfica de esta técnica.

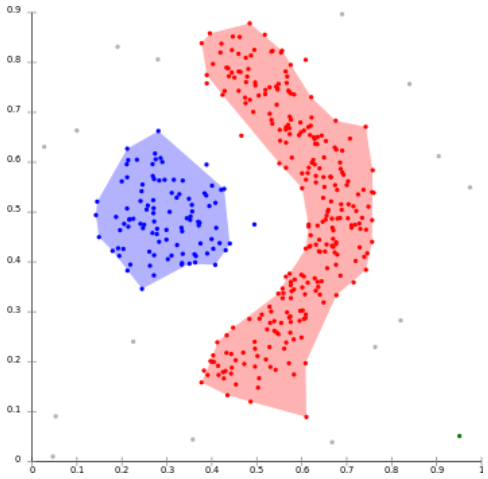

Ilustración 16: Distribución de datos en clusters utilizando DBSCAN

Para reducir el espacio de estados en el agente desarrollado en este proyecto con Q-Learning, se ha utilizado la combinación de dos algoritmos de agrupación para hallar el número de clusters que mejor se podía ajustar al conjunto de tuplas de experiencia. En primer lugar se ha ejecutado el algoritmo Expectation-Maximization, sin indicarle ningún cluster, de tal manera que ajustase un modelo por sí mismo. A continuación, se ha tomado como referencia inicial el número de clusters que ha generado el algoritmo anterior, de cara a ejecutar el algoritmo K-Medias para obtener los centroides de cada cluster, los cuales servirán para determinar a qué cluster pertenece el estado en el que se encuentra el agente durante la ejecución. Todo este proceso se ha realizado mediante el uso de Weka, un software Open Source de minería de datos desarrollado en Java por la Universidad de Waikato (Nueva Zelanda) que contiene una colección de algoritmos de aprendizaje automático y herramientas de regresión, clustering, pre-procesamiento de datos, reglas de asociación y visualización [12]. Para poder usar este software con las tuplas de experiencia es necesario añadir la siguiente cabecera al fichero:

- Relación de datos:

```
@relation <nombre de la relación>
```

- Declaración de las variables que componen las instancias del conjunto de datos:

```
@attribute <nombre del atributo> <tipo de dato>
```

- Declaración de las instancias y sus valores:

```
@data
<valor atributo 1>,<valor atributo 2>,…,<valor atributo n>
```

3.8 Q-Learning Aproximado

El método Q-Learning Aproximado es muy similar al Q-Learning anteriormente explicado pero con algunas diferencias. En este método no existe la Tabla Q, en su lugar el valor Q de cada par estado-acción se aproxima mediante una función lineal basada en las características que componen los estados y unos pesos asignados a cada una de estas características [9] [13]:

$$Q(s,a) = \sum_{i=1}^{n} f_i(s,a)\, w_i$$

- f = Característica (feature) del estado.
- w = Peso asociado a la característica.
- s = Estado.
- a = Acción.

A su vez, estos pesos están asociados a la acción que se ejecuta, por tanto este cálculo sería la multiplicación entre el vector de características asociadas al estado en el que nos encontramos por el vector de pesos (del mismo tamaño que el de características) asociado a la acción ejecutada. Teniendo esto en cuenta los pasos que sigue el método son similares a los de Q-Learning, solo que en este caso se van a entrenar los pesos: En primer lugar se inicializan los pesos a un valor númerico, después se toma una tupla de experiencia y se realiza una actualización sobre los pesos asociados a cada característica a partir de la acción y los datos contenidos en la tupla. Se repiten estos dos últimos pasos hasta que los valores de los pesos converjan, momento en el cual se habrá obtenido la política de acciones correcta. Para ello usa la siguiente función de actualización por cada peso [9] [13]:

$$w_i \rightarrow w_i + \alpha \cdot \left[\left(r + \gamma \max_{a'} Q(s',a')\right) - Q(s,a)\right] \cdot f_i(s,a)$$

- f = Característica (feature) del estado.
- w = Peso asociado a la característica.
- s = Estado inicial.
- a = Acción inicial.
- α = Tasa de aprendizaje.
- r = Refuerzo.
- γ = Tasa de descuento.
- s' = Estado siguiente.
- a' = Acción siguiente.

Como se puede observar, la función de actualización de este método es bastante similar a la función estocástica del Q-Learning tradicional, conservando las mismas tasas y el cálculo de la recompensa futura, pero con la diferencia de que ahora el valor acumulado en las sucesivas iteraciones del proceso de actualización recae sobre el peso, y se incorpora a su vez el valor de la característica asociada a éste.

La principal ventaja que ofrece este método respecto al algoritmo original es la escalabilidad sin la necesidad de ejecutar ningún algoritmo extra, como ocurriría en el caso de Q-Learning con clustering. Como ya se citó anteriormente, cada acción tiene asociado su propio vector de pesos lo que hace que el problema ahora sea independiente del tamaño del espacio de estados, permitiendo la adición de nuevas características más descriptivas para las distintas situaciones del problema. Esta consecuencia se debe a que en el proceso de actualización se toma el estado, sea cual sea, de la tupla que se esté leyendo en el momento y se le asigna un vector de pesos dependiendo únicamente de la acción que contiene esta. De este modo, si en la siguiente tupla se repite la misma acción pero no el estado, se vuelve a asignar de nuevo el mismo vector. Como resultado de todo este proceso se genera una tabla de pesos entrenados a partir de todos los estados contenidos en las tuplas, con tantas filas como el número de acciones del problema, y tantas columnas como el número de características que contienen los estados. Esta tabla será consultada durante la ejecución del agente desarrollado con este algoritmo para decidir la mejor acción a llevar a cabo, de forma que este multiplicará el estado en el que se encuentra por cada uno de los vectores contenidos en la tabla y elegirá la acción que más valor Q le reporte. Suponiendo un problema en el cual los estados contienen número alto de atributos, y la combinación de estos, por tanto, da lugar a una gran cantidad de estados, la tabla de pesos generada por Q-Learning Aproximado será en comparación muchísimo más pequeña que la tabla Q que resultaría del algoritmo de Q-Learning tradicional al depender sobre todo del espacio de acciones (el cual normalmente es más pequeño que el de estados), lo que supone, en estos casos, una gran mejoría en términos de viabilidad.

4 Marco legal

En este apartado se incluyen los términos legales de la licencia del software utilizado para el desarrollo del proyecto. Tanto el simulador MarioAI, como el programa Q-Learning, así como Weka, tienen una licencia de software libre Open Source [14], a la cual corresponde el siguiente marco legal:

"Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* *Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.*

* *Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.*

* *Neither the name of the MarioAI nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.*

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

5 Diseño e implementación de la solución técnica

En este apartado se describirá como se ha diseñado e implementado la solución al problema, describiendo la arquitectura del sistema desarrollado, la representación de los datos, estados y acciones, como se han tomado las tuplas de aprendizaje, y los distintos aspectos relacionados con el análisis y el diseño del software. También se hablará sobre las alternativas de diseño que se han propuesto o probado durante el desarrollo.

5.1 Arquitectura del sistema

El sistema desarrollado se compone de 4 módulos: Simulador de juego (Software MarioAI), Agente entrenador (preprogramado o humano), Módulo de aprendizaje (Software Qlearning) y Agente Automático. Estos módulos interactúan con el usuario y entre sí, generando un flujo de datos de entrada y salida representado en la Ilustración 17.

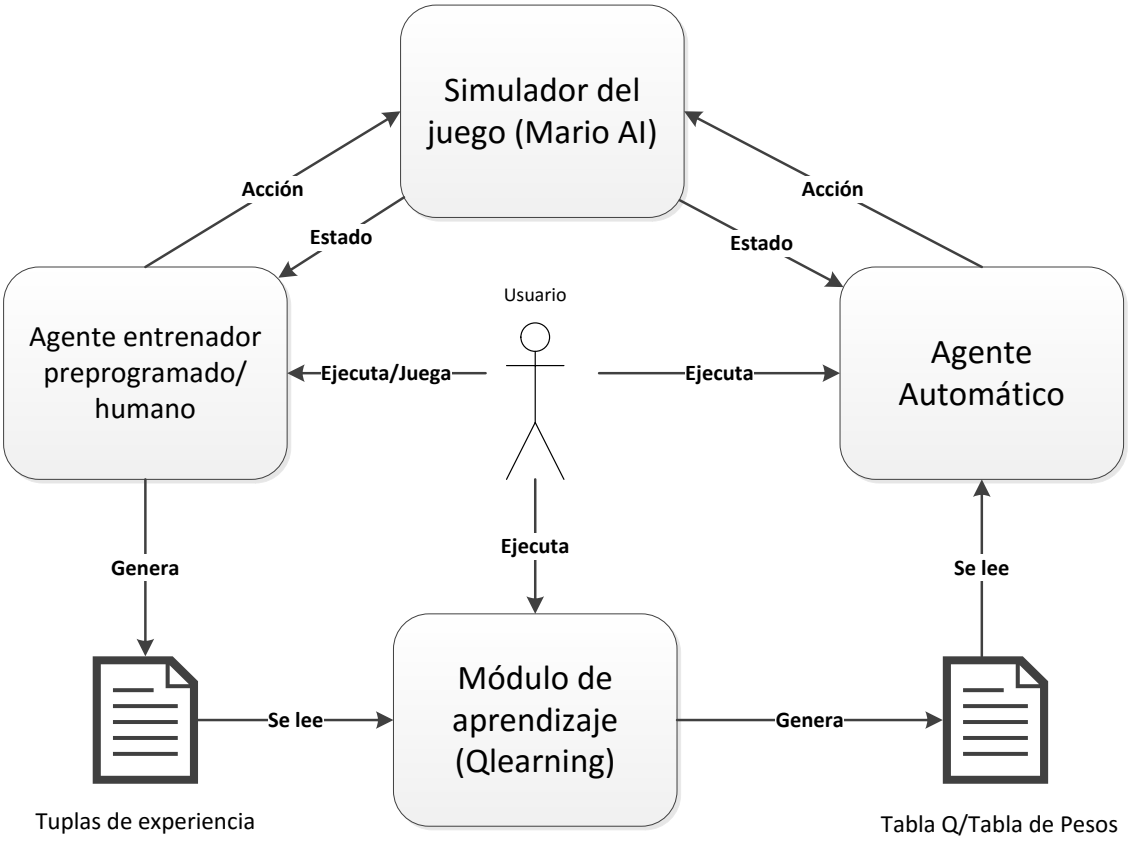

Ilustración 17: Diagrama con la arquitectura del sistema.

Las funciones e interacción de cada módulo son las siguientes:

- **Agente entrenador**: Este módulo será el encargado de generar las tuplas de experiencia para entrenar posteriormente el agente automático. Puede contener el software de un agente preprogramado, el cual ejecuta un número de acciones determinadas solo en aquellos casos que estén especificados en su código, o un agente humano, el cual permite a un humano jugar partidas usando el teclado y realizando acciones con total libertad.

  La ejecución de este módulo se lleva a cabo por el usuario, el cual deberá especificar mediante parámetros, al menos, el nombre del agente, la dificultad y la semilla del nivel. Una vez puesto en marcha, también se activará el simulador del juego, con el cual mantendrá un constante intercambio de datos. Este último, durante la partida, devolverá una matriz de datos al agente de la que se extraerá la información referente al estado de Mario, a partir del cual el agente o jugador decidirá llevar a cabo una determinada acción que mandará de vuelta al simulador. Cuando termine la ejecución se generará un fichero de texto con las tuplas generadas durante las partidas, las cuales serán aprovechadas por otros módulos del sistema.

- **Simulador del juego**: Este módulo se ejecuta junto con el de Agente preprogramado/humano o con el de Agente automático. Contiene todo el software referente al simulador del juego (MarioAI, ver El simulador: MarioAI (Infinite Mario Bros)) (motor gráfico, interfaz, controladores…) y procesa variables a partir de las cuales se puede extraer el estado de Mario y su puntuación. Durante su ejecución estará en constante comunicación con el agente que lo activó, recibiendo de este una acción y devolviendo por su parte el estado y la puntuación de Mario.

- **Módulo de aprendizaje**: Este módulo contiene el software QLearning (ver Software base: Q-Learning) en el cual están implementados los distintos algoritmos de Aprendizaje Automático utilizados para entrenar los agentes automáticos desarrollados. Al ser ejecutado por el usuario, este módulo recibe como parámetro de entrada el fichero con las tuplas de experiencia generadas por el Módulo Agente preprogramado/humano, las cuales serán procesadas por el algoritmo de aprendizaje que corresponda. Al final de la ejecución se devolverá un fichero de texto que contendrá la tabla aprendida por el algoritmo, el cual se reutilizará desde otros módulos.

- **Agente automático**: En este módulo se pondrá a prueba el agente automático entrenado. El proceso de ejecución es similar al del módulo Agente preprogramado/humano, manteniendo la comunicación con el simulador, pero en este caso la acción a llevar a cabo se decidirá desde la tabla generada por el Módulo de aprendizaje, la cual será un fichero de entrada para este módulo.

5.2 Diseño de la representación para Aprendizaje por Refuerzo

En este apartado se detallan cada una de las características utilizadas para la creación de los estados y que a su vez participan en la función Q, así como el espacio de acciones, la función de refuerzo y sus rangos de valores además de otros aspectos relevantes sobre los datos.

Todos los atributos que se detallan a continuación se han recogido utilizando una generalización de 1 (intermedia) sobre la matriz que devuelve el simulador (ver El simulador: MarioAI (Infinite Mario Bros)), y utilizando solo posiciones cercanas a Mario (3 casillas hacia arriba, 3 casillas hacia abajo, 3 hacia atrás y 3 hacia delante) tal y como se muestra en la Ilustración 18. Se ha decidido escoger tan solo estas casillas al considerar que representan la distancia más relevante a la hora de tomar decisiones en cada uno de los instantes de la partida y aportan una mejor información de la progresión de Mario en el estado siguiente. Estos instantes se contabilizan mediante ticks, los cuales representan el momento en el que se produce la interacción entre el simulador y el agente, incluyendo desde la observación del entorno hasta la emisión de la acción por este último.

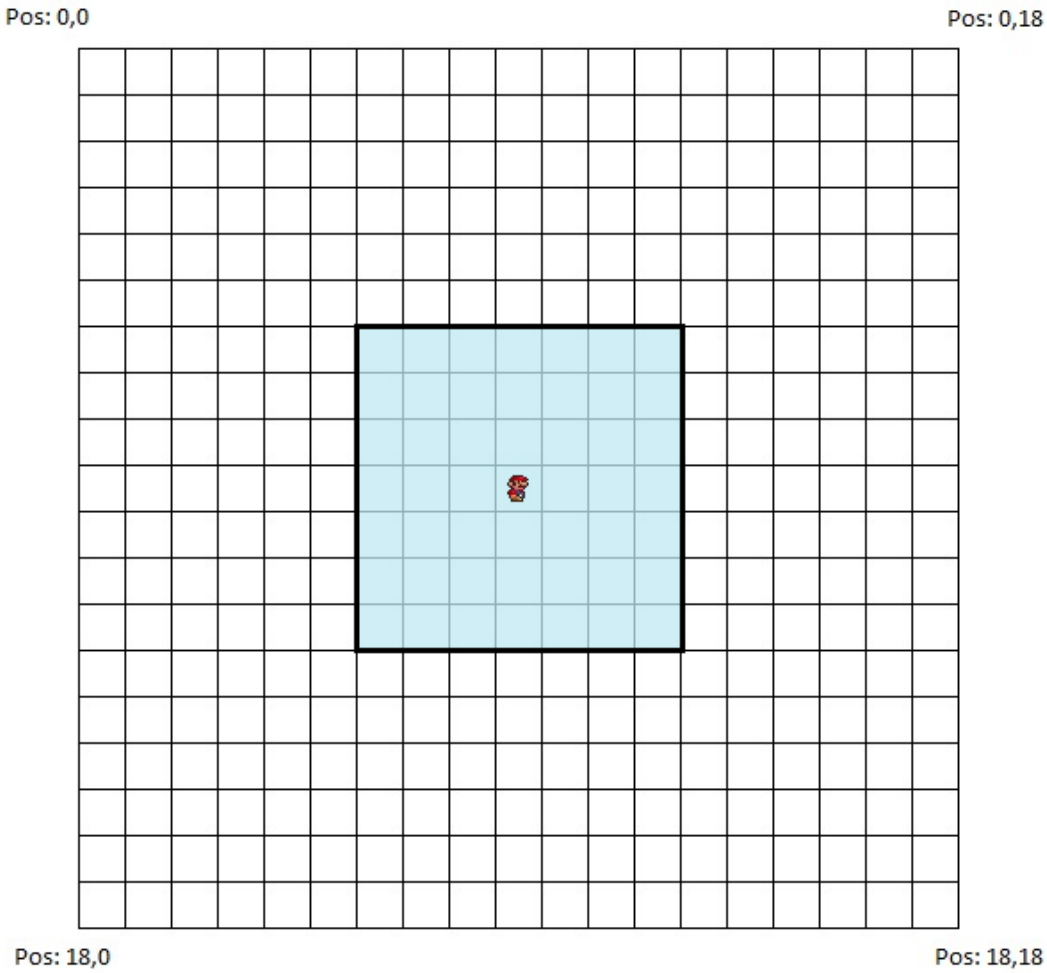

Ilustración 18: Espacio relevante para Mario (resaltado en azul) sobre la matriz de observación completa.

Los atributos que representan el estado siguiente conservan las mismas características que se presentan a continuación, pero a esta versión de los atributos se les marcará como "N+K" tras su nombre para distinguirlos del estado inicial. El intervalo escogido en el conjunto final de datos para separar ambos estados ha sido de 10 ticks, representando así la duración de la acción ejecutada en el estado inicial, de forma que queden reflejadas en el estado siguiente las consecuencias generadas por la misma.

5.2.1.1 Espacio de estados:

En este caso el tamaño del espacio de estados se correspondería con todas las combinaciones que pueden generar entre sí las 11 características que se describirán a continuación. Al tratarse en su mayoría de atributos con un rango de valores numéricos entre 0 y 1, el tamaño del espacio de estado que generan las posibles combinaciones entre ellos es infinito, lo que justifica el uso de Q-Learning Apróximado y Clustering para favorecer la viabilidad del aprendizaje.

El hecho de que muchas características presenten el citado rango de valores númericos se debe al uso de la distancia euclídea [15], la cual se deduce a partir del Teorema de Pitágoras y representa la distancia en línea recta que hay entre dos puntos pertenecientes a un espacio euclídeo. Dados dos puntos P y Q pertenecientes a un espacio euclídeo n-dimensional, la distancia euclídea entre ambos se definiría de forma general de la siguiente manera:

$$P = (p_1, p_2, \dots, p_n)$$
$$Q = (q_1, q_2, \dots, q_n)$$

$$d(P,Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

De forma más visual (Ilustración 19), suponiendo un par de puntos P_1 y P_2 con coordenadas cartesianas (x_1, y_1) y (x_2, y_2):

$$d(P_1, P_2) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

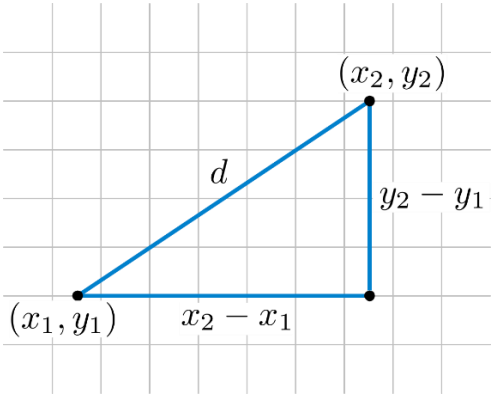

Ilustración 19: Representación gráfica del cálculo de la distancia euclídea. En esta se puede observar claramente la relación que guarda con el Teorema de Pitágoras.

Aplicada al problema de Super Mario Bros la distancia euclídea permite trazar la distancia que existe entre Mario y un elemento de la escena. Esta medida se aplica directamente sobre la porción de la matriz de la escena tomada del simulador, tomando como puntos X e Y la posición que ocupan en ella los objetos de la escena. Dichas coordenadas han de ser procesadas a continuación para que el cálculo de la distancia sea correcto, al no cumplir las condiciones del espacio euclídeo, por lo que se transformarán al espacio cartesiano, el cual sí las cumple, tomando la posición de Mario en la matriz como origen de coordenadas (ver Ilustración 20), mediante los siguientes cálculos:

$$posiciónMarioX = 9$$
$$posiciónMarioY = 9$$

$$xCartesiana = posiciónElementoX - posiciónMarioX$$
$$yCartesiana = posiciónMarioY - posiciónElementoY$$

- posiciónMarioX: columna en la que se encuentra Mario en la matriz devuelta por el sistema.
- posiciónMarioY: fila en la que se encuentra Mario en la matriz devuelta por el sistema.
- posiciónElementoX: columna en la que se encuentra un elemento en la matriz devuelta por el sistema.
- posiciónElementoY: fila en la que se encuentra un elemento en la matriz devuelta por el sistema.
- xCartesiana, yCartesiana = posición del elemento en el espacio cartesiano.

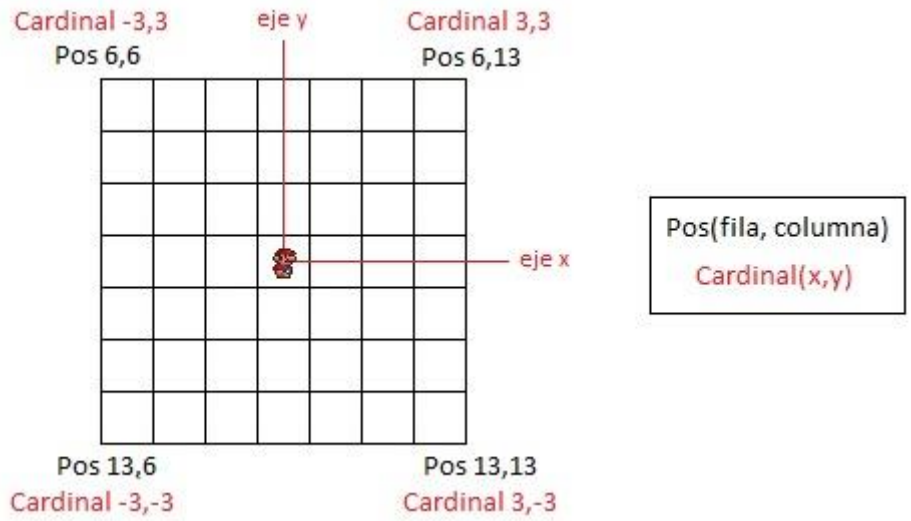

Ilustración 20: Transformación del espacio relevante a espacio cartesiano (euclídeo).

Para que los algoritmos aprendan a dar más peso a los elementos más cercanos a Mario se ha procedido a invertir las distancias, dividiendo la distancia euclídea máxima alcanzable en la matriz del espacio relevante (aplicando la fórmula este valor sería $\sqrt{(0-3)^2+(0-3)^2} = \sqrt{18} = 4.24$) entre la distancia al elemento (salvo que esta sea 0, en cuyo caso representa que no existe el elemento en la escena), de tal manera que los elementos más cercanos obtengan un valor mayor que los lejanos, permitiendo a los algoritmos aprender prioridades sobre ciertos elementos o acciones cuando la escena este repleta de objetos.

Los atributos que se han utilizado para generar los estados son los siguientes:

- **caminoDespejado**: Esta variable indica si Mario tiene el camino despejado de obstáculos, monedas, enemigos, bloques, setas o flores, ayudando a identificar de forma clara las mejores acciones que puede realizar cuando se da este caso. En el caso de Q-Learning Aproximado esta característica evita que el estado sea un vector de ceros cuando no haya elementos en escena, lo que impediría a dicho algoritmo aprender o ejecutar acciones en este caso. Tiene rango binario: tomará el valor 1 en el caso de que los citados elementos no estén presentes en la escena, y 0 en el caso de que si lo estén.

- **enemigos[Delante, Detrás]**: Estas variables indican si hay enemigos en las casillas posteriores a la posición de Mario, en el caso de enemigosDelante, o en las anteriores, en el caso de enemigosDetras, mediante un rango de valores entre 0 y 1 que expresa la distancia a la que está el enemigo: si el valor de la variable es igual a 0 indica que no hay enemigos en la escena o que están demasiado lejos como para ser relevantes; si el valor es distinto de 0 significa que hay un enemigo en el espacio relevante para Mario, tomando un valor cercano a 0 si está muy alejado, o próximo o igual a 1 si está muy cerca. Cabe destacar que ambas variables incluyen enemigos que estén delante o detrás de Mario aunque su posición sea más elevada o inferior a la de Mario y siempre toman la distancia al enemigo más cercano en el caso de que haya más de uno.

- **bloqueArriba**: Esta variable indica si hay bloques en las posiciones superiores a Mario, tanto por la izquierda como por la derecha, mediante un rango de valores entre 0 y 1 que expresa la distancia a la que está el bloque más cercano: si el valor de la variable es igual a 0 indica que no hay bloques en la escena o que están demasiado lejos como para ser relevantes; si el valor es distinto de 0 significa que hay un bloque en el espacio relevante para Mario, tomando un valor cercano a 0 si está muy alejado, o próximo o igual a 1 si está muy cerca. Gracias a esta variable el algoritmo puede aprender a romper los bloques para encontrar monedas y desbloquear flores o setas y, en ocasiones, incluso puede ayudar a Mario a variar su ruta para evitar bloqueos.

- **setaDetras**: Esta variable indica si hay una seta detrás, independientemente de si está arriba o debajo de Mario, mediante un rango de valores entre 0 y 1 que expresa la distancia a la que está la seta más cercana: si el valor de la variable es igual a 0 indica que no hay setas en la escena o que están demasiado lejos como

para ser relevantes; si el valor es distinto de 0 significa que hay una seta en el espacio relevante para Mario, tomando un valor cercano a 0 si está muy alejada, o próximo o igual a 1 si está muy cerca. En la matriz "Merged" o combinada, las setas y las monedas se codifican con el mismo identificador, lo que hace que se tengan que tratar, por lo general, del mismo modo. En este caso este tratamiento es contraproducente debido a que la ganancia que aporta al refuerzo el retroceder a por una moneda es mínima e incluso provoca que el algoritmo confunda cuando usar ciertas acciones. Sin embargo, recoger una seta que está detrás resulta positivo porque, aparte de aportar una jugosa puntuación, mejora la supervivencia de Mario y por tanto aumentan las posibilidades de superar el nivel. Debido a esto, esta variable hace uso de la matriz de enemigos ("Enemies"), de la cual puede extraer el código de la seta por separado, aislando así el caso descrito anteriormente.

- **monedaSeta[Arriba, Delante]**: Estas variables identifican cuando una moneda o una seta se encuentran en las posiciones superiores o posteriores a Mario en la matriz. A diferencia del caso que describe la característica "setaDetras", en esta situación se trata del mismo modo a monedas y a setas puesto que no se ve perjudicado al refuerzo y la forma de coger ambos elementos es la misma, por lo que en este caso no se provocan confusiones al algoritmo. Del mismo modo que otras variables, "monedaSeta[Arriba, Delante]", tiene un rango de valores entre 0 y 1 que expresa la distancia a la que se encuentra la moneda o la seta más cercana: si el valor de la variable es igual a 0 indica que no hay ni monedas ni setas en la escena o que están demasiado lejos como para ser relevantes; si el valor es distinto de 0 significa que hay una moneda o una seta en el espacio relevante para Mario, tomando un valor cercano a 0 si están muy alejadas, o próximo o igual a 1 si están muy cerca.

- **flor[Detrás, Arriba, Delante]**: Estas variables identifican cuando una flor se encuentra en las posiciones anteriores ("florDetras"), superiores ("florArriba") o posteriores ("florDelante") a Mario. Para ello utilizan un rango de valores entre 0 y 1, que al igual que muchas de las variables anteriormente descritas, expresa la distancia a la que se encuentra la flor más cercana: si el valor de la variable es igual a 0 indica que no hay flores en la escena o que están demasiado lejos como para ser relevantes; si el valor es distinto de 0 significa que hay una flor en el espacio relevante para Mario, tomando un valor cercano a 0 si está muy alejada, o próximo o igual a 1 si está muy cerca.

- **obstáculoDelante**: Esta variable indica si hay obstáculos en las posiciones de la matriz posteriores a Mario mediante un rango de valores entre 0 y 1, intervalo que, del mismo modo que muchas de las variables ya descritas, expresa la distancia al obstáculo más cercano: si el valor de la variable es igual a 0 indica que no hay obstáculos en la escena o que están demasiado lejos como para ser relevantes; si el valor es distinto de 0 significa que hay un obstáculo en el espacio relevante para Mario, tomando un valor cercano a 0 si está muy alejado, o próximo o igual a 1 si está muy cerca. Esta variable generaliza e incluye todos los

tipos de obstáculo que Mario puede encontrar en su camino como laderas, tuberías, cañones o ladrillos.

En el siguiente diagrama (Ilustración 21)se puede ver de forma gráfica el rango de visión que toma cada uno de los anteriores atributos dependiendo de si se refieren a elementos detrás, arriba o delante:

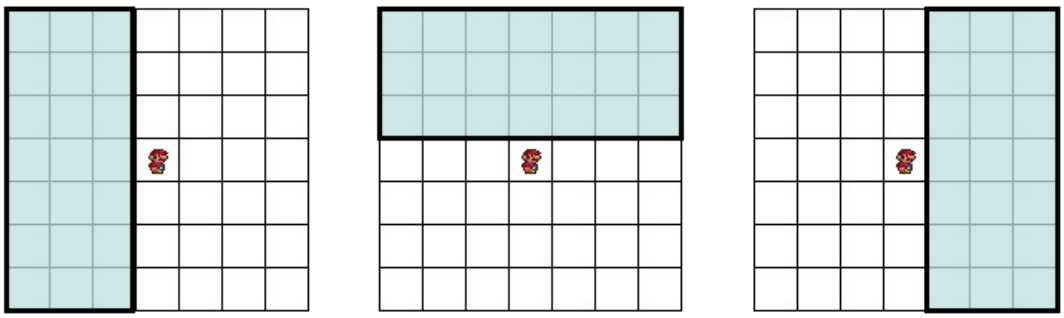

Ilustración 21: Rango de visión de Mario detrás, arriba y delante. Las casillas que ve Mario se indican en color azul.

5.2.1.2 Espacio de acciones:

El espacio de acciones de este problema se obtiene a partir de las posibles combinaciones que se pueden dar entre las teclas de acción del simulador, las cuales son:

- Disparar (Tecla A)
- Saltar (Tecla S)
- Ir hacia la izquierda (Flecha izquierda)
- Agacharse (Flecha abajo)
- Ir hacia la derecha (Flecha derecha)

Debido a la complejidad que tiene la acción Disparar (Tecla A) para ser aprendida por el algoritmo y el poco uso que tiene la acción Agacharse (Flecha abajo), ambas han sido descartadas de la codificación final. En función de si se pulsan o no estas teclas (true o false), se han codificado las 6 posibles acciones que puede realizar Mario mediante el siguiente rango de valores:

- **0 – Quieto**: {Tecla S = false, Flecha izquierda = false, Flecha derecha = false}.
- **1 – Avanza hacia delante**: {Tecla S = false, Flecha izquierda = false, Flecha derecha = true}.
- **2 – Retrocede**: {Tecla S = false, Flecha izquierda = true, Flecha derecha = false}.
- **3 – Salta**: {Tecla S = true, Flecha izquierda = false, Flecha derecha = false}.
- **4 – Avanza saltando**: {Tecla S = true, Flecha izquierda = false, Flecha derecha = true}.
- **5 – Retrocede saltando**: {Tecla S = true, Flecha izquierda = true, Flecha derecha = false}.

5.2.1.3 Función de refuerzo:

El resultado la función de refuerzo indica cómo de bien ha progresado Mario en una instancia de acuerdo a unos objetivos establecidos, los cuales pueden ser extraídos a partir de las características escogidas para formar los estados. Por ejemplo, en este proyecto, hay varias características que sitúan la posición de un enemigo, por tanto se puede deducir que uno de los objetivos del agente puede ser eliminar la mayor cantidad posible de estos. Lo mismo ocurre con otras características que sitúan algunos elementos de la escena como flores, monedas, o setas, por lo que de igual manera se deduce que el agente busca coger y descubrir todos los que pueda. Por otra parte, el agente también debe aprender a lograr estos objetivos sin recibir daño de los enemigos, por lo tanto, también se debería aplicar una penalización para cuando esto ocurra. Para cuantificar estos factores se ha decidido hacer uso de la puntuación interna del simulador, la cual engloba muy bien los objetivos propuestos para el proyecto, aplicándoles recompensas o penalizaciones en función de cómo juegue Mario. Dichas recompensas y penalizaciones son las siguientes:

- Mario se pasa el nivel: 1024 puntos.
- Recoger monedas: 16 puntos.
- Recoger flor de fuego: 64 puntos.
- Recibir daño: -42 puntos.
- Matar enemigos: 10 puntos.
- Recoger seta: 58 puntos.

Como se puede observar, la puntuación no incluye la distancia recorrida de Mario, con lo cual por sí sola no aportaría refuerzo a la simple acción de avanzar, la cual es fundamental para alcanzar el objetivo básico del juego: pasarse el nivel. Por tanto, a esta puntuación se le añade la distancia que Mario lleva recorrida en el nivel, obtenida a partir de una variable del simulador que suma 1 punto cada vez que da un paso adelante y resta 1 punto cada vez que retrocede. Teniendo en cuenta todos estos puntos, la función de refuerzo queda de la siguiente manera:

$$f(tick) = puntuación + distancia\ recorrida$$

Para obtener el refuerzo que produce la acción ejecutada en cada tupla de experiencia se realiza la diferencia entre esta función en el tick $n + k$ (estado siguiente, después de un intervalo de k ticks) y la función aplicada sobre el tick n (estado inicial de la tupla):

$$r = f(n + k) - f(n)$$

Cabe comentar que para algunas acciones se tienen llevan a cabo algunas modificaciones sobre la función de refuerzo para mejorar la forma en la que son cuantificadas. Es el caso de las acciones que retroceden, las cuales, según el planteamiento de la función de refuerzo, obtienen una recompensa negativa o muy baja a pesar de haber logrado el objetivo por el que fueron ejecutadas. Esto se soluciona anulando la aplicación de la distancia sobre la función de refuerzo en

aquellas acciones que han tenido éxito al retroceder, de tal manera, que solamente aparece en el resultado la puntuación obtenida al matar o recolectar objetos.

5.3 Alternativas de diseño de la representación

En este apartado se van a mencionar y razonar algunas características o representaciones de los estados que se han manejado durante el desarrollo del proyecto. Son las siguientes:

- **IsMarioOnGround**: Este atributo indica si Mario está en el suelo o no mediante el rango de valores 1-true y 0-false. Aporta información relevante sobre los saltos y rebotes que se realizan en los ejemplos, independientemente del agente que se haya usado para obtenerlos. Finalmente se descartó del proyecto y fue sustituida por la variable "caminoDespejado" (ver Diseño de la representación para Aprendizaje por Refuerzo), dado que no aprendía bien con el algoritmo Q-Learning Aproximado en aquellos casos en los que no había ningún elemento en la escena. Probablemente el uso combinado de la variable "caminoDespejado" con "isMarioOnGround" podría dar un buen resultado y, tal vez, permitiría nuevas combinaciones de acciones más óptimas y similares al comportamiento humano cuando Mario está en el aire, pero no se ha podido probar finalmente.

- **Rango de valores binario**: Las características diseñadas en los primeros modelos de representación de estados que se desarrollaron durante este proyecto tenían un rango binario (0,1). Este modelo funcionaba bastante bien con el algoritmo Q-Learning, pero sin embargo no terminaba de funcionar correctamente con Q-Learning Aproximado dado que no conseguía capturar en su totalidad el uso de ciertas acciones en determinados estados. Esto es debido a que al etiquetar con 0-false o 1-true, solo expresa la existencia o no de algunos de los elementos de la escena, lo que provocaba que, por ejemplo, en el caso de que existiese un enemigo detrás y un obstáculo delante, el algoritmo aprendiese que para saltar los obstáculos que están delante se tiene que llevar a cabo un salto hacia atrás por la sencilla razón de que ha obtenido más refuerzo al haber matado enemigos cercanos al obstáculo que saltándolo. Por esta razón se sustituyó este rango por el de distancias euclídeas de la versión definitiva, al describir mejor estos casos y ser también compatible con Q-Learning. Probablemente se podría haber llevado a cabo este proyecto con éxito usando este modelo mediante alguna modificación en el sistema de puntuaciones o refuerzo, pero no se ha podido probar dado lo costoso que resultaría realizar cambios en ese sistema y a la poca garantía que ofrece cambiar la función de refuerzo solo para este caso.

- **Uso de la matriz completa**: Otra de las opciones que se probó durante el desarrollo del proyecto fue que las características tomasen datos la matriz de observación en su totalidad en lugar de tomarlos solo del espacio cercano a Mario. Este diseño se planteó en sustitución de la variable "isMarioOnGround", anteriormente citada, antes de ser sustituida por la característica definitiva "caminoDespejado", para evitar el caso en el cual no hay elementos en el espacio

relevante para Mario. Este planteamiento se descartó tras comprobar que en este caso, pese al uso de la distancia euclídea, se daban conflictos en el aprendizaje similares a los del rango binario. Tal vez, la inclusión de más características que ayudasen a situar mejor los elementos del escenario y a establecer aún más prioridades sobre ellos haría que funcionase este modelo.

5.4 Creación de las tuplas de experiencia

Una vez diseñada la representación del problema, el siguiente paso a seguir es tomar tuplas de experiencia a partir de las cuales entrenar el agente final. Para tomar dichas tuplas se ejecuta en el sistema el ya mencionado **módulo Agente preprogramado/humano**, en el cual se aloja un agente con acciones programadas o un agente con un controlador para que un humano juegue partidas. El proceso de generación consiste llevar a cabo ejecuciones del agente contenido en el módulo en distintos niveles, de tal manera que pueda obtener el mayor número de experiencias posibles, y una vez se crea que hay suficientes ejemplos para las acciones contempladas por ese agente se cambia a otro con distinta programación para entrenar las acciones restantes. A continuación se detallan los agentes utilizados para generar los ejemplos finales del problema, así como la proporción en la que se han ejecutado y finalidad que persigue la programación cada uno:

- **BasicAAAgent.java**: Se trata de un agente preprogramado solo con las acciones avanzar y saltar hacia delante, las cuales ejecuta en caso de que haya enemigos u obstáculos delante. El uso de este agente proporciona una buena cantidad de ejemplos de avance, salto de obstáculos y esquivar enemigos, y además el hecho de que sea un agente no-humano elimina las posibles indecisiones o pulsaciones erróneas, o de más, que podría tener una persona al jugar, dando como resultado unos ejemplos muy claros. Este agente se ha ejecutado una vez por nivel en 10 niveles diferentes con dificultad 0.

- **P3Agent.java**: Este agente es el mismo que se desarrolló para la práctica final de la asignatura Aprendizaje Automático pero modificado para que obtenga la información que requiere este proyecto. Se trata de un agente automático basado en Q-Learning + Clustering que busca maximizar el avance y las muertes conseguidas a lo largo del nivel, con lo que proporcionará al agente de este proyecto ejemplos claros, al no estar controlado por un humano, de cómo matar enemigos delante y detrás además de añadir más ejemplos de avance y salto de obstáculos. Al igual que BasicAAAgent, este agente se ha ejecutado una vez por nivel en 10 niveles diferentes con dificultad 0.

- **HumanAgentTFG.java**: Se trata de un agente que implementa un controlador para que un humano sea el que juegue las partidas, capturando las teclas que pulsa para obtener la acción que realiza en cada momento. Debido a que un humano puede emitir acciones mucho más rápido que un agente preprogramado o automático y puede dudar a la hora de ejecutarlas, se ha sesgado mucho la lectura y el uso de este agente con la finalidad de que los ejemplos que emita sean lo más

claros posibles. De esta manera este agente se ha usado para obtener aquellas acciones que no cubren los otros agentes por ser muy complejas o por no estar programadas en ellos. Los ejemplos generados por este agente se centran principalmente en coger flores o champiñones (acción compleja), golpear bloques para obtener bonus y matar enemigos detrás. Al capturar solo ejemplos muy concretos, este agente ha sido ejecutado en 15 niveles con dificultad 0 para darles un peso relevante en el conjunto total de tuplas.

Como se puede observar se ha preferido el uso de agentes preprogramados o automáticos sobre el agente humano debido no solo a la ya mencionada claridad de los ejemplos sino también a que no hacen uso de la acción 0-Quieto. En los primeros ejemplos generados se hacía un mayor uso del agente humano para capturar la información, pero tras completar el proceso de entrenamiento al completo y ejecutar el agente automático producido el resultado obtenido difería mucho de lo esperado: el algoritmo, principalmente Q-Learning Aproximado, daba demasiado peso a la acción 0-Quieto cuando es la que menos aporta al objetivo del agente, generando un conflicto grave en su aprendizaje. Analizando los ejemplos se comprobó que, debido al intervalo de ticks entre los estados de las tuplas, se grababa la acción 0-Quieto y justo después se ejecutaba otra acción que producía los cambios en el estado siguiente y el refuerzo, los cuales no se correspondían con la acción real. Un ejemplo de este caso podría ser que en el estado del *tick n* se hallase un enemigo delante de Mario y la acción asociada al estado sea 0-Quieto, y sin embargo, en el siguiente, en lugar de haber recibido daño y seguir vivo enemigo, el enemigo aparece muerto, lo que demuestra que se ha tenido que ejecutar una acción intermedia para que este muriese. Esta es, por tanto, la razón de que se haya practicado un sesgo tan alto sobre el agente humano y apenas aparezcan ejemplos con la acción 0-Quieto.

5.5 Diseño del software desarrollado

En este apartado del proyecto se va analizar el problema principal, asignando una metodología adecuada para el desarrollo del mismo y dividiéndolo en subtareas más sencillas, con la finalidad de identificar aquellas condiciones y objetivos que debe cumplir de cara a su uso.

5.5.1 Metodología empleada

La metodología empleada para el desarrollo de este proyecto se corresponde con un **Modelo en Cascada**, el cual asegura una buena y estricta organización de las etapas de desarrollo ayudando así a que el software se complete con un alto nivel de calidad dentro del plazo establecido por el cliente. Este modelo fue propuesto por Winston Royce en 1970, siendo por tanto uno de los más longevos, y ha servido como modelo base para el resto de metodologías presentadas posteriormente. Presenta un ciclo de vida que admite iteraciones entre las fases, pero a su vez es el más rígido de todos: el software se debe realizar siguiendo una secuencia de fases en la que antes de pasar de una fase a otra se deben llevar a cabo varias revisiones para comprobar si se puede

avanzar en el proceso. Típicamente este modelo contiene las actividades que se muestran en la Ilustración 22 en su ciclo de vida [16] [17]:

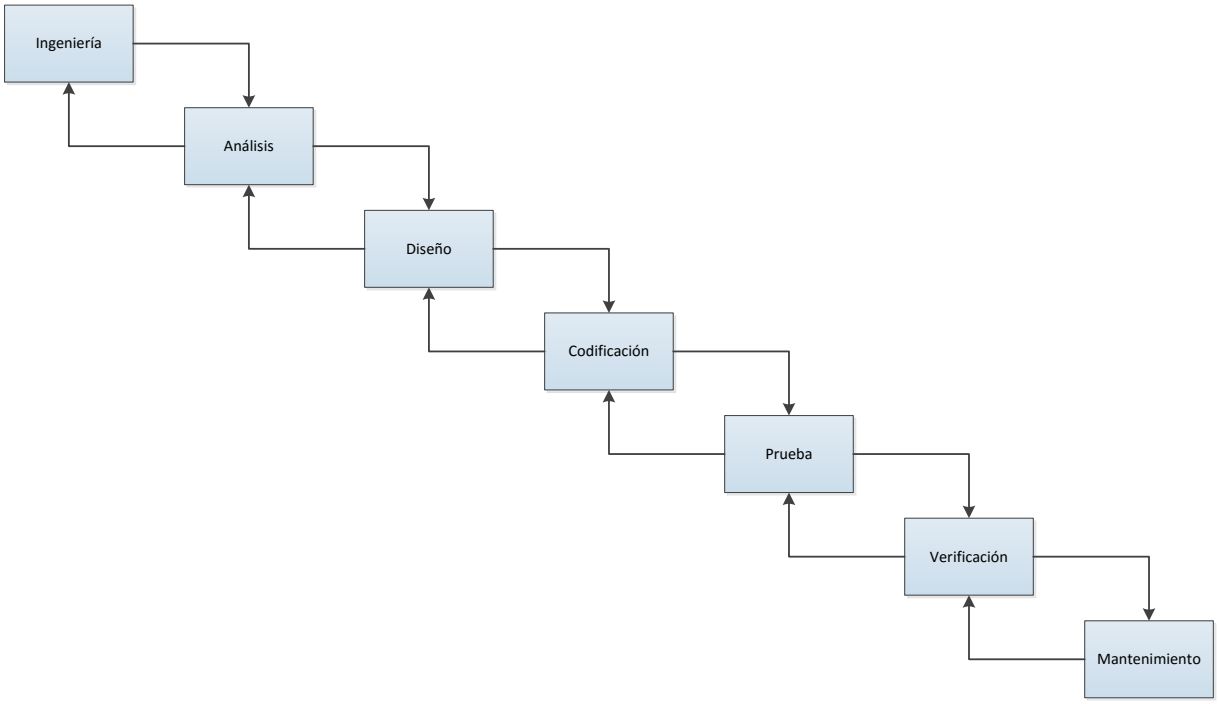

Ilustración 22: Fases del Modelo en Cascada

A continuación se detallará en que consiste cada una de estas actividades y cómo se ha adaptado a este proyecto:

- **Ingeniería**: En esta fase de la metodología se realiza un estudio preliminar del sistema para identificar los elementos que lo van a componer con el fin de extraer posteriormente los requisitos del software. En este proyecto esta fase se ha renombrado como **Estudio** y en ella se ha llevado a cabo un estudio de los distintos algoritmos y elementos de software que compondrán el sistema.

- **Análisis**: En esta fase se recopilan los requisitos que debe cumplir el software a desarrollar. De igual manera, la fase de **Análisis** de este proyecto se centra en recopilar los requisitos del usuario y de software, establece los casos de uso que debe cumplir el sistema, y además en ella traza la planificación y presupuesto del proyecto e inicia su documentación.

- **Diseño**: En esta fase se establece la arquitectura del software y se transforman los requisitos recogidos durante el análisis en una representación software previa a la codificación. En la fase de **Diseño** de este proyecto, de igual manera, se construye la arquitectura que cumplirá el sistema y se diseñan las clases que lo compondrán.

- **Codificación**: En la fase de codificación se transforma lo anteriormente diseñado en código legible para la máquina. En este proyecto la fase de codificación recibe el nombre de **Implementación** y en ella se escriben en lenguaje de software los distintos algoritmos necesarios para el funcionamiento del sistema.

- **Prueba**: En esta fase se prueba la lógica y funciones codificadas en la anterior fase del modelo, asegurando que la entrada de las mismas produce el resultado requerido. De igual manera en la fase de **Prueba** de este proyecto se diseñan diversas pruebas que verifican el funcionamiento de los distintos módulos del sistema.

- **Verificación**: En la fase de verificación se procede a entregar el software desarrollado al cliente para que este pueda finalmente ejecutarlo. En este proyecto esta fase recibe el nombre de **Revisión** y en ella se realiza además una última revisión a la documentación del software antes de ser entregada junto con el mismo.

- **Mantenimiento**: En esta fase se realizan los cambios que requiera el software después de la entrega al cliente, tales como corrección de errores, mejoras de rendimiento o adaptación del mismo a nuevos entornos (sistemas operativos u otros dispositivos). Esta fase queda fuera de la planificación de este proyecto por el momento, dado que a la hora de la entrega de este documento se encontrará en fase de **Revisión** (o **Verificación**).

5.5.2 Requisitos del sistema

Un requisito de sistema documenta e identifica unívocamente una condición o capacidad que se debe cumplir para la correcta resolución del problema así como para satisfacer las necesidades del usuario [16]. Se clasifican en dos niveles en función del enfoque y la fuente del requisito:

- **Requisitos de usuario:** Vienen dados por el usuario en un lenguaje natural y están enfocados a plantear el programa a partir de sus necesidades.

- **Requisitos de sistema:** Tienen su fuente en el analista del proyecto y están orientados a refinar el planteamiento del problema indicado en los requisitos de usuario, así como añadir ciertas capacidades o restricciones que debe cumplir el diseño.

5.5.2.1 Requisitos de usuario

Estos requisitos pueden ser de dos tipos: de **capacidad** (indican los objetivos que debe cumplir el sistema) y de **restricción** (describen cómo se tiene que alcanzar un objetivo del sistema). Constan de los siguientes apartados:

- **Identificador:** Código unívoco asignado a cada requisito. Tiene la siguiente forma: unas siglas que indican la categoría a la que pertenece el requisito, siendo US para los requisitos de usuario y SIS para los requisitos de sistema, y un número de tres cifras para diferenciar entre los requisitos que pertenecen a la misma categoría.
- **Nombre:** Nombre que toma el requisito.
- **Fuente:** Fuente de la que proviene el requisito.
- **Tipo:** Determina el tipo al que pertenece el requisito de usuario. Puede tomar los valores: “Capacidad” o “Restricción”.
- **Necesidad:** Determina la obligatoriedad del requisito. Puede tomar los siguientes valores: “Esencial”, “Deseable” y “Opcional”.
- **Prioridad:** Indica el nivel de importancia del requisito. Puede tomar los valores: “Alta”, “Media” y “Baja”.
- **Descripción:** Descripción breve, clara y detallada sobre el requisito.

US-001	
Nombre	Selección de agente
Fuente	Usuario
Tipo	Capacidad
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario podrá seleccionar el agente que desea ejecutar.

Tabla 1: Requisito de Usuario US-001

US-002	
Nombre	Selección de nivel
Fuente	Usuario
Tipo	Capacidad
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario podrá seleccionar el nivel a ejecutar y su dificultad.

Tabla 2: Requisito de Usuario US-002

US-003	
Nombre	Visualización de la partida
Fuente	Usuario
Tipo	Capacidad
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario podrá ver como se realiza la partida en la interfaz del simulador al ejecutar un agente seleccionado.

Tabla 3: Requisito de Usuario US-003

US-004	
Nombre	Jugar partidas automáticas
Fuente	Usuario
Tipo	Capacidad
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario podrá jugar partidas de forma automática.

Tabla 4: Requisito de Usuario US-004

US-005	
Nombre	Jugar partidas manuales
Fuente	Usuario
Tipo	Capacidad
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario podrá jugar partidas de forma manual.

Tabla 5: Requisito de Usuario US-005

US-006	
Nombre	Entrenar un agente automático
Fuente	Usuario
Tipo	Capacidad
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario podrá entrenar un agente automático a partir de tuplas de experiencia.

Tabla 6: Requisito de Usuario US-006

US-007	
Nombre	Mostrar acciones por pantalla
Fuente	Usuario
Tipo	Capacidadl
Necesidad	Deseable
Prioridad	Media
Descripción	El usuario podrá ver por pantalla las acciones que se van ejecutando durante la partida.

Tabla 7: Requisito de Usuario US-007

US-008	
Nombre	Mostrar resultados de partida por pantalla
Fuente	Usuario
Tipo	Capacidad
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario podrá ver al finalizar la partida los resultados obtenidos en la partida.

Tabla 8: Requisito de Usuario US-008

US-009	
Nombre	Mostrar resultados de entrenamiento por pantalla
Fuente	Usuario
Tipo	Capacidad
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario podrá ver al finalizar la partida los resultados obtenidos en el entrenamiento.

Tabla 9: Requisito de Usuario US-009

US-010	
Nombre	Claridad en la interfaz de la partida
Fuente	Usuario
Tipo	Restricción
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario debe ver claramente que sucede en todo momento en la partida, así como obtener una presentación clara de los resultados de la misma.

Tabla 10: Requisito de Usuario US-010

US-011	
Nombre	Claridad en la interfaz del entrenamiento
Fuente	Usuario
Tipo	Restricción
Necesidad	Esencial
Prioridad	Alta
Descripción	El usuario debe obtener una presentación clara de los resultados del proceso de entrenamiento.

Tabla 11: Requisito de Usuario US-011

US-012	
Nombre	Compatibilidad con Windows
Fuente	Usuario
Tipo	Restricción
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema debe ser compatible con Windows 7, 8 y 10.

Tabla 12: Requisito de Usuario US-012

US-013	
Nombre	Lenguaje de desarrollo
Fuente	Usuario
Tipo	Restricción
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema debe estar desarrollado en Java.

Tabla 13: Requisito de Usuario US-013

5.5.2.2 Requisitos de sistema

Estos requisitos pueden ser de dos tipos: **funcionales** (indican las funciones y operaciones que se debe realizar el sistema para alcanzar un objetivo) y **no funcionales** (indican las restricciones impuestas sobre cómo se tiene que alcanzar un objetivo) [16]. Constan de los siguientes apartados:

- **Identificador:** Código unívoco asignado a cada requisito. Tiene la siguiente forma: unas siglas que indican la categoría a la que pertenece el requisito, siendo US para los requisitos de usuario y SIS para los requisitos de sistema, y un número de tres cifras para diferenciar entre los requisitos que pertenecen a la misma categoría.
- **Nombre:** Nombre que toma el requisito.
- **Fuente:** Fuente de la que proviene el requisito.
- **Tipo:** Determina el tipo al que pertenece el requisito de usuario. Puede tomar los valores: “Funcionales” o “No Funcionales”.
- **Necesidad:** Determina la obligatoriedad del requisito. Puede tomar los siguientes valores: “Esencial”, “Deseable” y “Opcional”.
- **Prioridad:** Indica el nivel de importancia del requisito. Puede tomar los valores: “Alta”, “Media” y “Baja”.
- **Descripción:** Descripción breve, clara y detallada sobre el requisito.

SIS-001	
Nombre	Inicialización de la partida
Fuente	Analista
Tipo	Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema permitirá al usuario seleccionar, mediante el uso de parámetros, el agente a ejecutar, el nivel de dificultad y la semilla a ejecutar.

Tabla 14: Requisito de Sistema SIS-001

SIS-002	
Nombre	Visualización de la partida
Fuente	Analista/Usuario
Tipo	Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema permitirá observar cómo se realiza la partida a través de la interfaz del simulador.

Tabla 15: Requisito de Sistema SIS-002

SIS-003	
Nombre	Visualización de la acción ejecutada
Fuente	Analista/Usuario
Tipo	Funcional
Necesidad	Deseable
Prioridad	Media
Descripción	Durante la ejecución de la partida el sistema imprimirá por pantalla la acción que selecciona el agente en cada momento de la misma.

Tabla 16: Requisito de Sistema SIS-003

SIS-004	
Nombre	Visualización del resumen de partida
Fuente	Analista/Usuario
Tipo	Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	Al finalizar la ejecución de la partida el sistema imprimirá por pantalla un resumen con los resultados de la partida.

Tabla 17: Requisito de Sistema SIS-004

SIS-005	
Nombre	Visualización del resultado del entrenamiento
Fuente	Analista/Usuario
Tipo	Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	Al finalizar la ejecución del proceso de entrenamiento el sistema imprimirá por pantalla la política de movimientos generada por el algoritmo.

Tabla 18: Requisito de Sistema SIS-005

SIS-006	
Nombre	Rendimiento en la ejecución de partidas
Fuente	Analista
Tipo	No Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	La ejecución de la partida propuesta por el usuario debe mantenerse estable hasta su finalización.

Tabla 19: Requisito de Sistema SIS-006

SIS-007	
Nombre	Claridad en la interfaz de la partida
Fuente	Analista/Usuario
Tipo	No Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema imprimirá por pantalla las acciones ejecutadas y los resultados haciendo uso tabulaciones y otros símbolos que garanticen la claridad de lo que se muestra.

Tabla 20: Requisito de Sistema SIS-007

SIS-008	
Nombre	Claridad en la interfaz del entrenamiento
Fuente	Analista/Usuario
Tipo	No Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema imprimirá por pantalla la política generada por el algoritmo seleccionado haciendo uso tabulaciones y otros símbolos que garanticen la claridad de lo que se muestra.

Tabla 21: Requisito de Sistema SIS-008

SIS-009	
Nombre	Sencillez de la interfaz
Fuente	Analista
Tipo	No Funcional
Necesidad	Deseable
Prioridad	Media
Descripción	Los comandos para la ejecución de los distintos niveles y agentes deben ser sencillos e intuitivos.

Tabla 22: Requisito de Sistema SIS-009

SIS-010	
Nombre	Comunicación con otros módulos del sistema
Fuente	Analista
Tipo	No Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema será capaz de comunicar el software Q-Learning con el simulador Mario AI para llevar a cabo tanto la ejecución y entrenamiento de los agentes.

Tabla 23: Requisito de Sistema SIS-010

SIS-011	
Nombre	Almacenaje de los datos de partida
Fuente	Analista
Tipo	Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema será capaz de recolectar datos de las partidas que juega y almacenarlos en un fichero de texto con formato Weka.

Tabla 24: Requisito de Sistema SIS-011

SIS-012	
Nombre	Almacenaje de los datos de entrenamiento
Fuente	Analista
Tipo	Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema será capaz de almacenar la política generada durante el entrenamiento en un fichero de texto con formato Weka.

Tabla 25: Requisito de Sistema SIS-012

SIS-013	
Nombre	Elección de la acción a ejecutar por un agente automático
Fuente	Analista
Tipo	Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El agente será capaz de determinar la acción a ejecutar más adecuada a partir del estado en el que se encuentre la partida.

Tabla 26: Requisito de Sistema SIS-013

SIS-014	
Nombre	Elección de la acción a ejecutar en agente manual
Fuente	Analista
Tipo	Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	Durante la ejecución de la partida el agente permitirá al usuario escoger, mediante el uso del teclado, una acción o una combinación de las siguientes acciones: retroceder, avanzar, saltar, disparar o agacharse.

Tabla 27: Requisito de Sistema SIS-014

SIS-015	
Nombre	Compatibilidad con el sistema operativo Windows
Fuente	Analista/Usuario
Tipo	No Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema podrá ser ejecutado en Windows 7, 8 o 10 sin ninguna incompatibilidad.

Tabla 28: Requisito de Sistema SIS-015

SIS-016	
Nombre	Lenguaje de programación Java
Fuente	Analista/Usuario
Tipo	No Funcional
Necesidad	Esencial
Prioridad	Alta
Descripción	El sistema estará desarrollado en su totalidad en el lenguaje de programación Java.

Tabla 29: Requisito de Sistema SIS-016

5.5.3 Casos de Uso

Los casos de uso documentan y especifican las diversas situaciones que se puede encontrar un actor al interactuar con un determinado programa, describiendo de esta manera la funcionalidad del mismo. En este proyecto solo habría un actor, el usuario, el cual ejecutará los posibles escenarios del programa tal y como se puede ver en el siguiente diagrama:

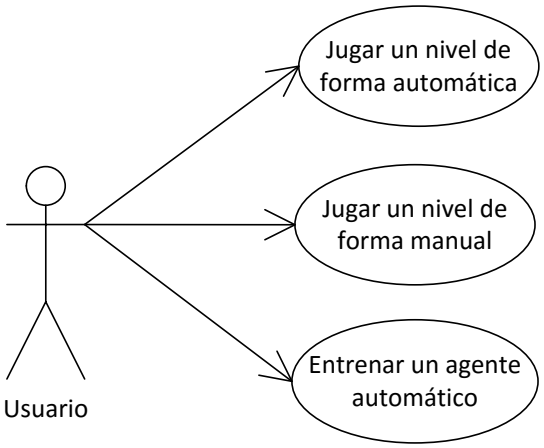

Ilustración 23: Diagrama de Casos de Uso

Para describir todo lo anteriormente mencionado, los casos de uso constan de los siguientes apartados:

- **Identificador:** Código unívoco asignado a cada caso de uso. Este tendrá la siguiente forma: Las siglas “CU”, que identificarán el elemento como un caso de uso, y un número de tres cifras para diferenciar a cada caso de uso de los demás.
- **Nombre:** Nombre asignado al caso de uso.
- **Descripción:** Descripción de la acción a detallar en el caso de uso.
- **Actores:** Actores que participan en el caso de uso.
- **Precondiciones:** Condiciones que se deben cumplir antes de ejecutar el caso de uso.
- **Postcondiciones:** Condiciones que se deben cumplir tras ejecutar el caso de uso.
- **Pasos:** Acciones que realiza el actor mientras se ejecuta el caso de uso.

CU-001	
Nombre	Jugar de forma automática.
Descripción	Jugar un nivel de forma automática con unos determinados parámetros.
Actores	Usuario.
Precondiciones	Abrir la consola y acceder a la carpeta del programa.
Postcondiciones	Obtención por pantalla de los resultados del nivel.
Pasos	1- El usuario escribe en la consola “ejecutar”. 2- En la misma línea y separado por un espacio, el usuario escribe el nombre del agente automático o preprogramado con el que desea ejecutar el juego. 3- A continuación del nombre del agente el usuario escribe el número de nivel y el número de semilla separados por un espacio. 4- Seguido al número de semilla, el usuario puede añadir parámetros adicionales si lo desea, tales como la longitud del nivel, tipos de enemigos o el modo inmortal de Mario. 5- El usuario ejecuta el programa con los parámetros elegidos y visualiza la partida.

Tabla 30: Caso de uso CU-001

CU-002	
Nombre	Jugar de forma manual.
Descripción	Jugar un nivel de forma manual con unos determinados parámetros.
Actores	Usuario.
Precondiciones	Abrir la consola y acceder a la carpeta del programa.
Postcondiciones	Obtención por pantalla de los resultados del nivel.
Pasos	1- El usuario escribe en la consola “ejecutar”. 2- En la misma línea y separado por un espacio, el usuario escribe el nombre del agente con el que desea ejecutar el juego. 3- A continuación del nombre del agente el usuario escribe el número de nivel y el número de semilla separados por un espacio. 4- Seguido al número de semilla, el usuario puede añadir parámetros adicionales si lo desea, tales como la longitud del nivel, tipos de enemigos o el modo inmortal de Mario. 5- El usuario ejecuta el programa con los parámetros elegidos y juega la partida utilizando las teclas de movimiento y acción.

Tabla 31: Caso de uso CU-002

CU-003	
Nombre	Entrenar un agente automático.
Descripción	Genera una política de movimientos para las características del agente a partir de las tuplas de experiencia recogidas.
Actores	Usuario.
Precondiciones	Disponer de tuplas de experiencia (formato Weka .arff).
Postcondiciones	Obtención del fichero que contiene la tabla de pesos.
Pasos	1- El usuario abre el programa mainMarioTFG.java (Q-Learning Aproximado) o mainMario.java (Q-Learning) con un editor de textos. 2- El usuario escribe en el programa la ruta donde se aloja el fichero con las tuplas de experiencia. 3- El usuario escribe en el programa la ruta donde desea alojar el fichero a generar con la tabla de pesos. 4- El usuario ejecuta el programa.

Tabla 32: Caso de uso CU-003

5.5.4 Matrices de trazabilidad

En este apartado se incluyen las matrices necesarias para verificar la trazabilidad y la relación existente entre los distintos requisitos y casos de uso expuestos con anterioridad:

- **Matriz de trazabilidad entre Requisitos de Usuario (filas) y Requisitos de Sistema (columnas)**. Aquellos requisitos que guarden relación entre sí tendrán una X en su casilla correspondiente:

	SIS-001	SIS-002	SIS-003	SIS-004	SIS-005	SIS-006	SIS-007	SIS-008	SIS-009	SIS-010	SIS-011	SIS-012	SIS-013	SIS-014	SIS-015	SIS-016
US-001	X															
US-002	X															
US-003		X														
US-004											X		X			
US-005											X			X		
US-006												X				
US-007			X													
US-008				X												
US-009					X											
US-010							X									
US-011								X								
US-012															X	
US-013																X

Tabla 33: Matriz de trazabilidad entre Requisitos de Usuario y Requisitos de Sistema

- **Matriz de trazabilidad entre Requisitos de Usuario (filas) y Casos de Uso (columnas)**. Aquellos Requisitos de Usuario y Casos de Uso que guarden relación tendrán una X en su casilla correspondiente.:

	CU-001	CU-002	CU-003
US-001	X	X	
US-002	X	X	
US-003	X	X	
US-004	X		
US-005		X	
US-006			X
US-007	X	X	
US-008	X	X	
US-009			X
US-010	X	X	
US-011			X
US-012			
US-013			

Tabla 34: Matriz de trazabilidad entre Requisitos de Usuario y Casos de Uso

5.5.5 Diseño de las clases

En este apartado se van a describir las clases que contienen los distintos módulos que componen el sistema. Para ilustrarlas de forma legible y clara en primer lugar se muestra un diagrama UML que establece las relaciones entre las distintas clases desarrolladas y, a continuación, se irá clase por clase describiendo sus atributos y funciones más relevantes.

5.5.5.1 Diagrama UML

En el siguiente diagrama (Ilustración 24) se muestran las relaciones existentes entre las distintas clases que componen el sistema:

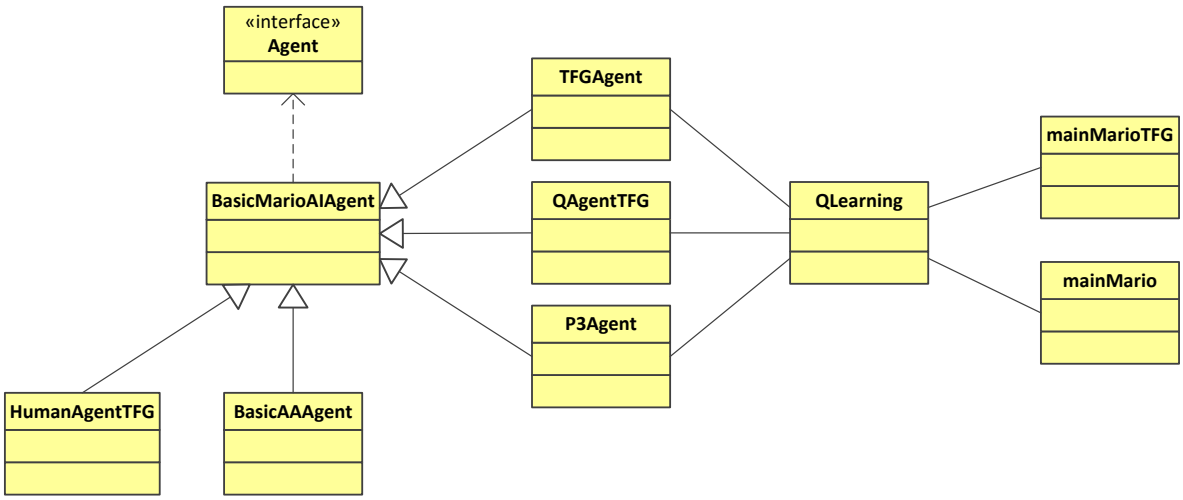

Ilustración 24: Diagrama UML de relación de clases.

A continuación, en los siguientes apartados, se detallan cada una de las clases que componen el diagrama, explicando cada una de sus funciones y atributos.

5.5.5.2 Interfaz Agent

Esta interfaz, perteneciente al módulo del Simulador de juego (Mario AI), deja planteada la funcionalidad de la que dispondrán los agentes del simulador.

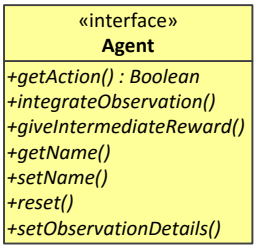

Ilustración 25: Interfaz Agent

La funcionalidad de los métodos que contiene esta interfaz serán explicados en las secciones correspondientes a las clases que la implementan.

5.5.5.3 Clase BasicMarioAIAgent

Esta clase, perteneciente al módulo del Simulador de juego (MarioAI), implementa la interfaz Agente y funciona a modo de clase padre para todos los agentes pertenecientes a los módulos de sistema Agente entrenador preprogramado/humano y Agente automático.

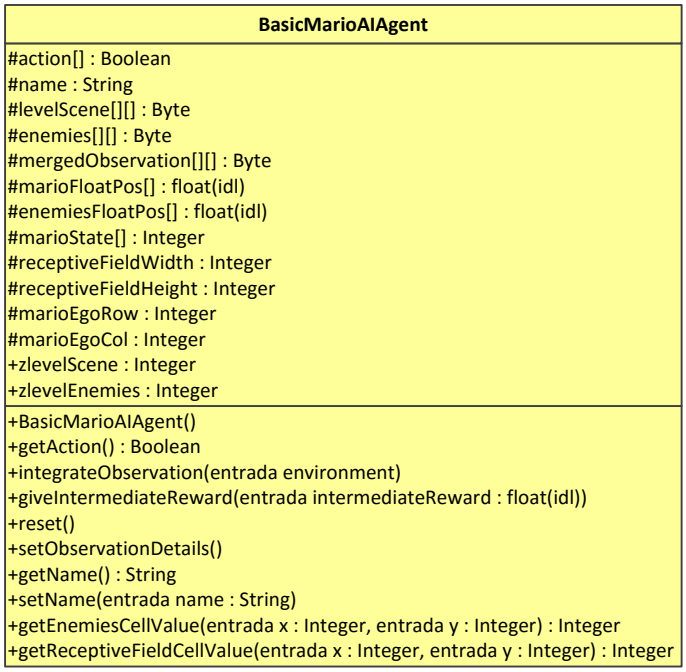

Ilustración 26: Clase BasicMarioAIAgent

El objetivo de esta clase es implementar los atributos y métodos necesarios para la funcionalidad planteada por la interfaz Agente. Debido a la implementación tan básica de los métodos y los atributos en esta clase y a que han sido diseñados previamente por los autores del simulador, estos se explicarán en las clases que heredan de esta, las cuales los especifican de acuerdo al problema propuesto en el proyecto.

5.5.5.4 Clase HumanAgentTFG

La clase HumanAgentTFG pertenece al módulo Agente entrenador preprogramado/humano, hereda de la clase BasicMarioAIAgent e implementa la interfaz Agent. El propósito del diseño de esta clase es dar la capacidad a un humano para jugar partidas de las que extraer información con la que entrenar el agente automático final. Por esta razón implementa los siguientes atributos y métodos:

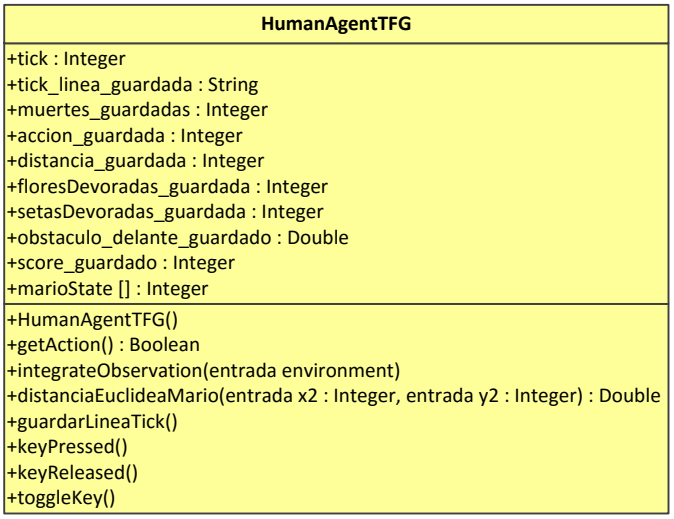

Ilustración 27: Clase HumanAgentTFG

Atributos:

- **tick**: Este atributo de tipo Integer se irá incrementando según avanza la partida, representando cada instante del juego.
- **tick_linea_guardada**: Este atributo de tipo String es el encargado de almacenar el estado y la acción inicial, de tal forma que pueda ser recuperado, tras el intervalo de ticks, para agregarle el estado siguiente y el refuerzo y conformar así la tupla de experiencia completa.
- Las variables **Muertes_guardadas, acción_guardada, distancia_guardada, floresDevoradas_guardada, setasDevoradas_guardada, obstáculo_delante_guardado** y **score_guardado** almacenan distintos datos sobre la partida, de tal manera que puedan ser usados, tras el intervalo de ticks, para hacer modificaciones sobre el refuerzo o la impresión de las tuplas.
- **marioState:** Es un array de tipo Integer obtenido desde el simulador MarioAI que contiene diferentes datos relevantes sobre el estado de Mario como, por ejemplo, si está en el suelo, si puede saltar, si está en modo grande….

Métodos:

- **HumanAgentTFG():** Este método es el constructor de clase en el que se inicializarán todas las variables descritas anteriormente y se utilizará la función reset() de la superclase, la cual reinicia las acciones que se hayan podido pulsar en anteriores partidas.
- **getAction():** Esta función devuelve al simulador la acción elegida por el agente para ser ejecutada.
- **integrateObservation():** En este método se procede a la lectura y extracción de datos de la matriz obtenida del entorno de la partida (environment), el cual recibe por parámetro, en cada tick o instante del juego. Dentro de este están declaradas

como variable todos los atributos del problema (descritos en Diseño de la representación para Aprendizaje por Refuerzo), los cuales obtendrán su valor correspondiente a partir de los bucles que recorren la matriz.

- **distanciaEuclideaMario():** Este método se encarga de calcular la distancia euclídea que existe desde un punto de la escena hasta Mario. Para ello recibe por parámetro la columna y la fila donde se encuentra el elemento, transforma estos valores en coordenadas cartesianas y calcula la distancia, devolviéndola al finalizar el método.
- **guardarLineaTick():** Esta función es la encargada de realizar la impresión a fichero de las tuplas generadas por el agente. En ella a su vez se incluye una cabecera de datos para poder ser procesado por Weka.
- Las funciones **keyPressed(), keyReleased()** y **toggleKey()**, son variables prediseñadas por los autores del simulador MarioAI, las cuales recogen y gestionan las pulsaciones que realiza el jugador humano en su teclado y las convierten en acciones para Mario.

5.5.5.5 Clase BasicAAAgent

La clase BasicAAAgent pertenece al módulo Agente entrenador preprogramado/humano, hereda de la clase BasicMarioAIAgent e implementa la interfaz Agent. El diseño de esta clase tiene como propósito ejecutar partidas utilizando solo las acciones preprogramadas de avanzar y saltar hacia delante, dependiendo de si hay peligro o no, y extraer información a partir de la cual entrenar el agente automático final. Por esta razón implementa los siguientes atributos y métodos:

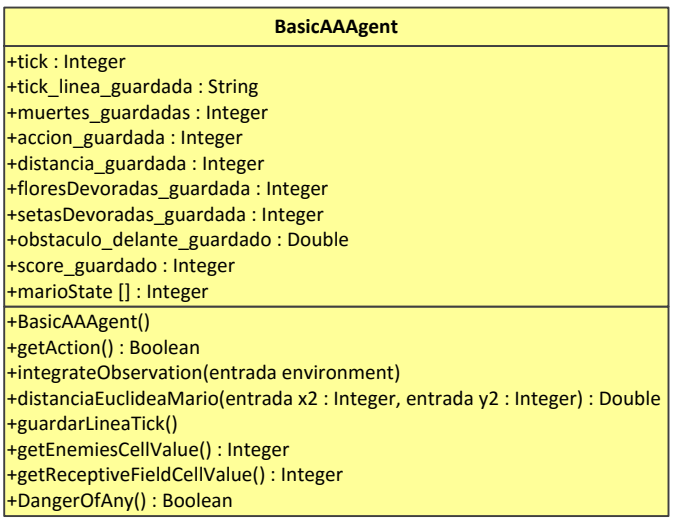

Ilustración 28: Clase BasicAAAgent

Atributos:

- **tick**: Este atributo de tipo Integer se irá incrementando según avanza la partida, representando cada instante del juego.
- **tick_linea_guardada**: Este atributo de tipo String es el encargado de almacenar el estado y la acción inicial, de tal forma que pueda ser recuperado, tras el intervalo de ticks, para agregarle el estado siguiente y el refuerzo y conformar así la tupla de experiencia completa.
- Las variables **muertes_guardadas, acción_guardada, distancia_guardada, floresDevoradas_guardada, setasDevoradas_guardada, obstáculo_delante_guardado** y **score_guardado** almacenan distintos datos sobre la partida, de tal manera que puedan ser usados, tras el intervalo de ticks, para hacer modificaciones sobre el refuerzo o la impresión de las tuplas.
- **marioState:** Es un array de tipo Integer obtenido desde el simulador MarioAI que contiene diferentes datos relevantes sobre el estado de Mario como, por ejemplo, si está en el suelo, si puede saltar, si está en modo grande….

Métodos:

- **BasicAAAgent():** Este método es el constructor de clase en el que se inicializarán todas las variables descritas anteriormente y se utilizará la función reset() de la superclase, la cual reinicia las acciones que se hayan podido pulsar en anteriores partidas.
- **getAction():** Esta función devuelve al simulador la acción elegida por el agente para ser ejecutada.
- **integrateObservation():** En este método se procede a la lectura y extracción de datos de la matriz obtenida del entorno de la partida (environment), el cual recibe por parámetro, en cada tick o instante del juego. Dentro de este están declaradas como variable todos los atributos del problema (descritos en Diseño de la representación para Aprendizaje por Refuerzo), los cuales obtendrán su valor correspondiente a partir de los bucles que recorren la matriz.
- **distanciaEuclideaMario():** Este método se encarga de calcular la distancia euclídea que existe desde un punto de la escena hasta Mario. Para ello recibe por parámetro la columna y la fila donde se encuentra el elemento, transforma estos valores en coordenadas cartesianas y calcula la distancia, devolviéndola al finalizar el método.
- **guardarLineaTick():** Esta función es la encargada de realizar la impresión a fichero de las tuplas generadas por el agente. En ella a su vez se incluye una cabecera de datos para poder ser procesado por Weka.
- Las funciones **getEnemiesCellValue(), getReceptiveFieldValue()** y **DangerOfAny()** son variables prediseñadas por los autores del simulador MarioAI que analizan la escena en busca de enemigos y obstáculos, y ejecutan una acción en función de lo observado.

5.5.5.6 Clase P3Agent

La clase P3Agent pertenece al módulo Agente entrenador preprogramado/humano, hereda de la clase BasicMarioAIAgent e implementa la interfaz Agent. Esta clase hace uso del algoritmo Q-Learning para decidir cuál es la mejor acción a realizar en cada instante de la partida, maximizando el avance y las muertes conseguidas. Al igual que en las demás clases de este módulo, en esta se extrae información a partir de la decisión que toma este agente con la finalidad de entrenar el agente automático final. Por esta razón implementa los siguientes atributos y métodos:

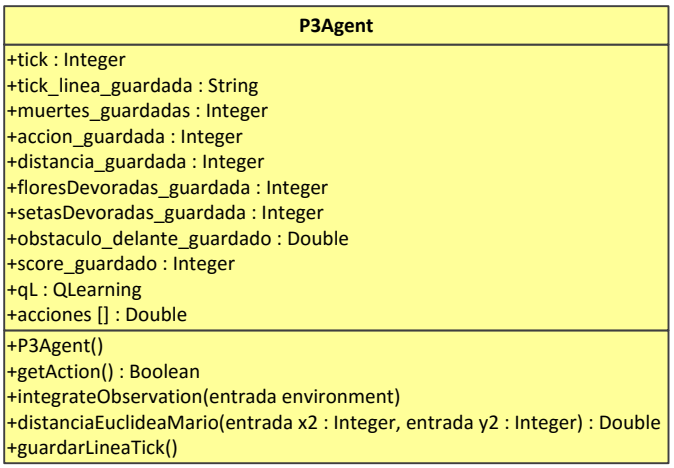

Ilustración 29: Clase P3Agent

Atributos:

- **tick**: Este atributo de tipo Integer se irá incrementando según avanza la partida, representando cada instante del juego.
- **tick_linea_guardada**: Este atributo de tipo String es el encargado de almacenar el estado y la acción inicial, de tal forma que pueda ser recuperado, tras el intervalo de ticks, para agregarle el estado siguiente y el refuerzo y conformar así la tupla de experiencia completa.
- Las variables **muertes_guardadas, acción_guardada, distancia_guardada, floresDevoradas_guardada, setasDevoradas_guardada, obstáculo_delante_guardado** y **score_guardado** almacenan distintos datos sobre la partida, de tal manera que puedan ser usados, tras el intervalo de ticks, para hacer modificaciones sobre el refuerzo o la impresión de las tuplas.
- **marioState:** Es un array de tipo Integer obtenido desde el simulador MarioAI que contiene diferentes datos relevantes sobre el estado de Mario como, por ejemplo, si está en el suelo, si puede saltar, si está en modo grande….
- **qL:** Instancia de la clase QLearning, necesaria para poder ejecutar el algoritmo que decidirá la acción a tomar por el agente.

Métodos:

- **P3Agent():** Este método es el constructor de clase donde se inicializarán todas las variables descritas anteriormente y se utilizará la función reset() de la superclase, la cual reinicia las acciones que se hayan podido pulsar en anteriores partidas. A su vez, también se inicializarán aquí los parámetros necesarios para la instancia de la clase QLearning y se cargará la Tabla Q asociada al agente.
- **getAction():** Esta función devuelve al simulador la acción elegida por el agente para ser ejecutada.
- **integrateObservation():** En este método se procede a la lectura y extracción de datos de la matriz obtenida del entorno de la partida (environment), el cual recibe por parámetro, en cada tick o instante del juego. Dentro de este están declaradas como variable todos los atributos del problema (descritos en Diseño de la representación para Aprendizaje por Refuerzo), los cuales obtendrán su valor correspondiente a partir de los bucles que recorren la matriz. Por otra parte, en esta función se hace una llamada al método obtenerMejorAcción, perteneciente a la clase Q-Learning, que decidirá la mejor acción a ejecutar según la información obtenida en el instante de juego.
- **distanciaEuclideaMario():** Este método se encarga de calcular la distancia euclídea que existe desde un punto de la escena hasta Mario. Para ello recibe por parámetro la columna y la fila donde se encuentra el elemento, transforma estos valores en coordenadas cartesianas y calcula la distancia, devolviéndola al finalizar el método.
- **guardarLineaTick():** Esta función es la encargada de realizar la impresión a fichero de las tuplas generadas por el agente. En ella a su vez se incluye una cabecera de datos para poder ser procesado por Weka.

5.5.5.7 Clase TFGAgent

La clase TFGAgent pertenece al módulo Agente automático, hereda de la clase BasicMarioAIAgent e implementa la interfaz Agent. Se trata del agente automático resultante de todo el proceso de entrenamiento llevado a cabo en el sistema en el que se ejecuta una acción decidida mediante el uso del algoritmo Q-Learning Aproximado. Esta clase implementa los siguientes atributos y métodos:

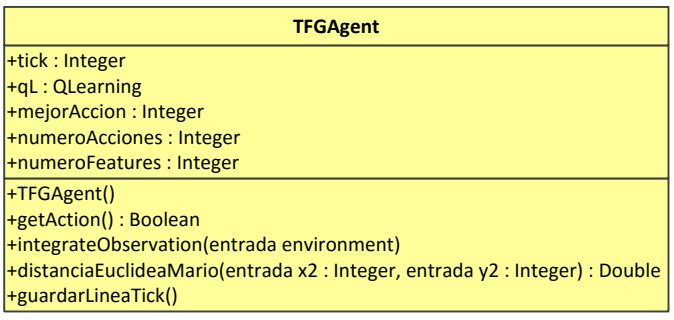

Ilustración 30: Clase TFGAgent

Atributos:

- **tick**: Este atributo de tipo Integer se irá incrementando según avanza la partida, representando cada instante del juego.
- **qL:** Instancia de la clase QLearning, necesaria para poder ejecutar el algoritmo que decidirá la acción a tomar por el agente.
- **numeroAcciones:** Define el número de acciones del problema.
- **numeroFeatures:** Define el número de características que componen los estados del problema.

Métodos:

- **TFGAgent():** Este método es el constructor de clase donde se inicializarán todas las variables descritas anteriormente y se utilizará la función reset() de la superclase, la cual reinicia las acciones que se hayan podido pulsar en anteriores partidas. A su vez, también se inicializarán aquí los parámetros necesarios para la instancia de la clase QLearning, que contiene el algoritmo de Q-Learning Aproximado, y se cargará la tabla de pesos asociada al agente.
- **getAction():** Esta función devuelve al simulador la acción elegida por el agente para ser ejecutada.
- **integrateObservation():**En este método se procede a la lectura y extracción de datos de la matriz obtenida del entorno de la partida (environment), el cual recibe por parámetro, en cada tick o instante del juego. Dentro de este están declaradas como variable todos los atributos del problema (descritos en Diseño de la representación para Aprendizaje por Refuerzo), los cuales obtendrán su valor correspondiente a partir de los bucles que recorren la matriz. Por otra parte, en esta función se hace una llamada al método mejorAccionQAprox, perteneciente a la clase Q-Learning, que decidirá la mejor acción a ejecutar según la información obtenida en el instante de juego.
- **distanciaEuclideaMario():** Este método se encarga de calcular la distancia euclídea que existe desde un punto de la escena hasta Mario. Para ello recibe por parámetro la columna y la fila donde se encuentra el elemento, transforma estos valores en coordenadas cartesianas y calcula la distancia, devolviéndola al finalizar el método.
- **guardarLineaTick():** Esta función es la encargada de realizar la impresión a fichero de las tuplas generadas por el agente. En ella a su vez se incluye una cabecera de datos para poder ser procesado por Weka.

5.5.5.8 Clase QAgentTFG

La clase QAgente pertenece al módulo Agente automático, hereda de la clase BasicMarioAIAgent e implementa la interfaz Agent. Es una variación de la clase TFGAgent: utiliza Q-Learning para decidir la mejor acción en lugar de la versión con aproximación del algoritmo. Esta clase implementa los siguientes atributos y métodos:

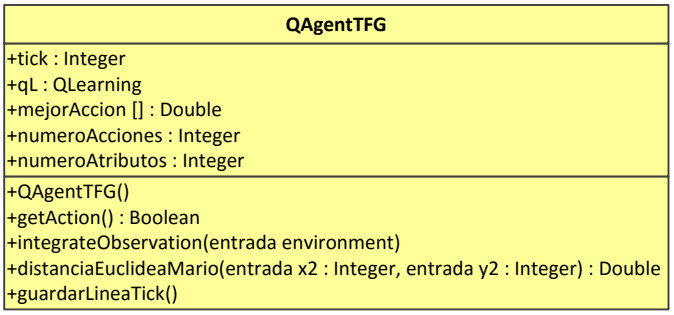

Ilustración 31: Clase QAgentTFG

Atributos:

- **tick**: Este atributo de tipo Integer se irá incrementando según avanza la partida, representando cada instante del juego.
- **qL:** Instancia de la clase QLearning, necesaria para poder ejecutar el algoritmo que decidirá la acción a tomar por el agente.
- **numeroAcciones:** Define el número de acciones del problema.
- **numeroAtributos:** Define el número de atributos que componen los estados del problema.

Métodos:

- **TFGAgent():** Este método es el constructor de clase donde se inicializarán todas las variables descritas anteriormente y se utilizará la función reset() de la superclase, la cual reinicia las acciones que se hayan podido pulsar en anteriores partidas. A su vez, también se inicializarán aquí los parámetros necesarios para la instancia de la clase QLearning y se cargará la Tabla Q asociada al agente.
- **getAction():** Esta función devuelve al simulador la acción elegida por el agente para ser ejecutada.
- **integrateObservation():**En este método se procede a la lectura y extracción de datos de la matriz obtenida del entorno de la partida (environment), el cual recibe por parámetro, en cada tick o instante del juego. Dentro de este están declaradas como variable todos los atributos del problema (descritos en Diseño de la representación para Aprendizaje por Refuerzo), los cuales obtendrán su valor correspondiente a partir de los bucles que recorren la matriz. Por otra parte, en esta función se hace una llamada al método obtenerMejorAccion, perteneciente a

la clase Q-Learning, que decidirá la mejor acción a ejecutar según la información obtenida en el instante de juego.

- **distanciaEuclideaMario():** Este método se encarga de calcular la distancia euclídea que existe desde un punto de la escena hasta Mario. Para ello recibe por parámetro la columna y la fila donde se encuentra el elemento, transforma estos valores en coordenadas cartesianas y calcula la distancia, devolviéndola al finalizar el método.
- **guardarLineaTick():** Esta función es la encargada de realizar la impresión a fichero de las tuplas generadas por el agente. En ella a su vez se incluye una cabecera de datos para poder ser procesado por Weka.

5.5.5.9 Clase QLearning

La clase QLearning pertenece al Módulo de Aprendizaje (QLearning), y contiene todos los métodos necesarios para llevar a cabo la implementación de los algoritmos Q-Learning y Q-Learning Apróximado. Mediante esta clase es posible entrenar el agente final, generando la Tabla Q o de pesos necesaria para su ejecución, y hacer que este tome decisiones en tiempo real llamando a sus métodos. En ella se implementan los siguientes atributos y funciones:

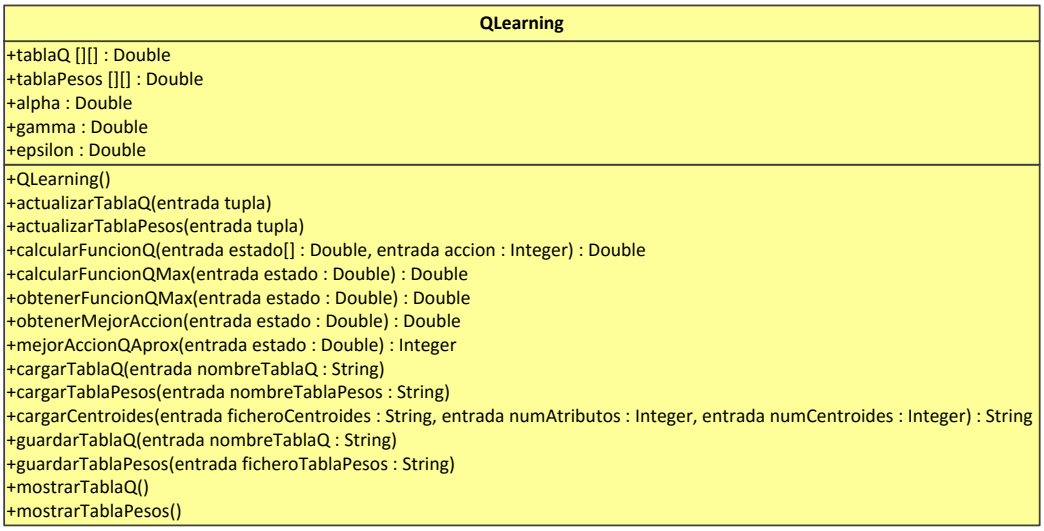

Ilustración 32: Clase Q-Learning

Atributos:

- **tablaQ**: Matriz de dos dimensiones destinada a almacenar la Tabla Q resultante de la ejecución del algoritmo Q-Learning.
- **tablaPesos**: Matriz de dos dimensiones destinada a almacenar la tabla de pesos resultante de la ejecución del algoritmo Q-Learning Aproximado.
- **alpha**: Variable que contiene la tasa de aprendizaje de los algoritmos Q-Learning y Q-Learning Aproximado.

- **gamma**: Variable que contiene la tasa de descuento de los algoritmos Q-Learning y Q-Learning Aproximado.
- **epsilon**: Parámetro adicional en el caso de que se vaya a usar una estrategia de exploración E-Greedy. En este proyecto, este atributo está en desuso.

Métodos:

- **QLearning():** Este método es el constructor de clase donde se inicializarán todas las variables descritas anteriormente en función del algoritmo que se ejecute: Q-Learning o Q-Learning Aproximado.
- **actualizarTablaQ():** Esta función recibe por parámetro una tupla de experiencia y actualiza la Tabla Q, aplicando el algoritmo de actualización de Q-Learning, con la información contenida en ella.
- **actualizarTablaPesos():** Esta función recibe por parámetro una tupla de experiencia y actualiza la tabla de pesos, aplicando el algoritmo de actualización de Q-Learning Aproximado, con la información contenida en ella.
- **calcularFuncionQ():** Este método calcula y devuelve el valor de la función Q de Q-Learning Aproximado a partir de unas características y una acción pasadas por parámetro.
- **calcularFuncionQMax():** Esta función recorre los vectores de pesos asignados a cada acción y devuelve el máximo valor que alcanza la funcion Q de Q-Learning Aproximado para unas features pasadas por parámetro.
- **obtenerFuncionQMax():** Esta función obtiene y devuelve el máximo valor de la función Q de Q-Learning para el estado pasado por parámetro.
- **obtenerMejorAccion():** Esta función obtiene y devuelve la mejor acción correspondiente al máximo valor de la función Q de Q-Learning.
- **mejorAccionQAprox():** Obtiene la mejor acción a partir de unas características pasadas en fase de ejecución usando el algoritmo Q-Learning Aproximado.
- **cargarTablaQ():** Carga la Tabla Q contenida en un fichero pasado por parámetro.
- **cargarTablaPesos():** Carga la tabla de pesos contenida en un fichero pasado por parámetro.
- **cargarCentroides():** Carga los centroides contenidos en un fichero por parámetro. Además utiliza el resto de parámetros para transponer la "matriz" de centroides que devuelve Weka y que esta pueda ser procesada por el programa.
- **guardarTablaQ():** Guarda la Tabla Q generada por el algoritmo Q-Learning en un fichero especificado por parámetro.
- **guardarTablaPesos():** Guarda la tabla de pesos generada mediante Q-Learning Aproximado en un fichero especificado por parámetro.
- **mostrarTablaQ():** Muestra por pantalla la Tabla Q generada por el algoritmo Q-Learning.

- **mostrarTablaPesos():** Muestra por pantalla la tabla de pesos generada mediante Q-Learning Aproximado.

5.5.5.10 Clase MainMario

La clase mainMario pertenece al Módulo de Aprendizaje (QLearning), y ejerce de clase principal para dicho módulo, leyendo las tuplas de aprendizaje y ejecutando los métodos relacionados con el algoritmo Q-Learning. Mediante esta clase se genera la Tabla Q necesaria para el funcionamiento del agente automático final. En ella están implementados los siguientes atributos y métodos:

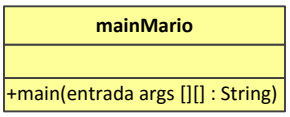

Ilustración 33: Clase mainMario

Atributos:

No hay, están contenidos dentro del método main().

Métodos:

- **main():** En este método se leen las tuplas de experiencia, pasadas como parámetro a la clase, se ajustan los parámetros del algoritmo Q-Learning y se llaman a los métodos de la clase Q-Learning asociados a este, los cuales se ejecutan con la información contenida en los ejemplos.

5.5.5.11 Clase mainMarioTFG

La clase mainMarioTFG pertenece al Módulo de Aprendizaje (QLearning), ejerce de clase principal para dicho módulo y es una variación de la clase mainMario para la ejecución de Q-Learning Aproximado. Esta clase lee las tuplas de aprendizaje y ejecuta los métodos relacionados con dicho algoritmo, generando la tabla de pesos necesaria para el funcionamiento del agente automático final. En ella están implementados los siguientes atributos y métodos:

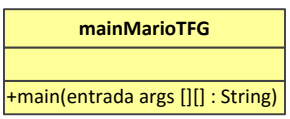

Ilustración 34: Clase MainMarioTFG

Atributos:

No hay, están contenidos dentro del método main().

Métodos:

- **main():** En este método se leen las tuplas de experiencia, pasadas como parámetro a la clase, se ajustan los parámetros del algoritmo Q-Learning Aproximado y se llaman a los métodos de la clase Q-Learning asociados a este, los cuales se ejecutan con la información contenida en los ejemplos.

6 Evaluación y Resultados

En este apartado se va a evaluar el agente desarrollado en sus dos versiones, con Q-Learning y con Q-Learning Aproximado. En primer lugar se establecerá y se razonará el criterio de evaluación a seguir, se definirán las pruebas y se realizará una observación de la evolución del aprendizaje. Finalmente se expondrán y se analizarán los resultados obtenidos que determinarán cuál es el agente con el funcionamiento más óptimo.

6.1 Descripción del entorno de evaluación

En los siguientes apartados se van a definir todos los criterios que se han seguido para realizar la evaluación del agente final.

6.1.1 ¿Qué se evalúa?

El propósito principal de esta evaluación es analizar el funcionamiento y la progresión del agente desarrollado en las versiones Q-Learning y Q-Learning en cada uno de los niveles que se han seleccionado como conjunto de pruebas. Para ello, no solo se establecerá una comparación entre ambas versiones para observar qué algoritmo se comporta mejor, sino que además estos agentes se enfrentarán a un agente preprogramado, lo que permitirá extraer los beneficios que aporta el uso de técnicas de Inteligencia Artificial en un problema frente a aquellas soluciones que no las incorporan. Por tanto se han seleccionado los siguientes agentes:

- Agente con Q-Learning: **QAgentTFG**.
- Agente con Q-Learning Aproximado: **TFGAgent**.
- Agente preprogramado: **BasicAAAgent** (Agente que ejecuta sólo las acciones avanzar y saltar hacia delante).

6.1.2 ¿Qué se va a medir en la evaluación?

El algoritmo Q-Learning, como se mencionó anteriormente, tiene como meta buscar una política de movimientos que resuelvan un problema **maximizando el refuerzo**, por tanto, para medir cómo de bien se comportan los agentes desarrollados habrá que evaluar aquellos aspectos que comprenden su refuerzo. En este problema el refuerzo escogido ha sido la suma de la puntuación, que engloba las muertes conseguidas y las flores, monedas y setas recogidas, y la distancia, por lo que será necesario observar el resultado que ha logrado el agente sobre estos objetivos al final de la partida. Para resolver esta necesidad se hace uso de la variable **Weighted Fitness**, implementada en el simulador, que calcula la puntuación y la distancia total conseguida al final de partida, ponderándola por el tiempo restante para superar el nivel. Dicho valor representa todos los objetivos a medir por lo que se escoge como valor de medición.

6.1.3 ¿Cómo se va realizar la evaluación?

La evaluación va a consistir en poner a prueba a los tres agentes seleccionados en distintos niveles del juego y comparar el resultado que obtiene cada uno de ellos sobre el valor de medición. Estos agentes correrán sobre el nivel 0 de dificultad del simulador, misma dificultad con la que han sido entrenados, y sobre las 10 primeras semillas del mismo.

6.1.4 Entrenamiento

Los agentes desarrollados realizan un aprendizaje de manera offline, por tanto, disponen de una fase de entrenamiento antes de ser ejecutados. En esta fase las tuplas de experiencia generadas previamente (ver Creación de las tuplas de experiencia) son leídas por el algoritmo para producir la tabla de pesos o la tabla Q, dependiendo de cuál se utilice, que usarán más tarde estos agentes para decidir sus acciones. Los parámetros y características que se han usado en este proceso para generar las tablas finales de cada agente se describen a continuación.

6.1.4.1 Agente Q-Learning Aproximado

En el caso del agente desarrollado con Q-Learning Aproximado la elección de las tuplas definitivas para el entrenamiento se ha llevado a cabo iterando, un gran número de veces, en la generación de tuplas hasta lograr un conjunto representativo del problema que se adaptase bien al algoritmo y generase buenos resultados. A su vez, durante dichas iteraciones también se han realizado ajustes sobre los ciclos de entrenamiento (número de veces que se lee todo el conjunto de tuplas) y las tasas de descuento y de aprendizaje del algoritmo, los cuales permiten controlar los valores que alcanzan los pesos durante el proceso, ayudando a su convergencia. Finalmente, con la elección de las tuplas definitivas, los valores de estos parámetros han quedado de la siguiente manera:

- **Número de ciclos de entrenamiento**: 3
- **Tasa de aprendizaje (α)**: 0.1
- **Tasa de descuento (γ)**: 0.9

6.1.4.2 Agente Q-Learning

En el caso del agente desarrollado con Q-Learning el proceso de selección de tuplas para el entrenamiento se ha llevado a cabo de forma similar al de Q-Learning Aproximado, pero en este caso, además de elegir el conjunto de entrenamiento correcto, también hay que determinar el número de clusters a los que se tiene que reducir el espacio de estados que genera los atributos de las tuplas. Para ello se hace uso del software Weka y los siguientes criterios:

- **Espacio de acciones**: Se supone que puede haber un número de clusters igual al número de acciones, de tal manera que cada cluster o estado represente cuando usar cada acción. El espacio de acciones del problema es de tamaño 6, pero por otra parte, una de las acciones (0-Quieto) no tiene uso, por lo que podría considerarse también de tamaño 5. Debido a este motivo se probarán distribuciones de 5 y 6 clusters.

- **Algoritmo Expectation Maximization (EM)**: Mediante el uso de Weka se ejecuta el algoritmo Expectation Maximization (ver Clustering), que devolverá una posible distribución de clusters sobre el conjunto de tuplas seleccionado. Tras probar con distintas semillas e iteraciones el resultado se ha mantenido siempre en una distribución de 3 clusters.

- **Número de atributos**: En este caso se supone que cada estado represente cada una de las metas marcadas por los atributos, por lo que el número de clusters será igual al espacio de estados: 11.

Estas posibles distribuciones se han probado en Weka con el algoritmo K-Medias y se ha medido la puntuación obtenida por cada una, según el valor de medición establecido (Weighted Fitness), en cada una de las diez primeras semillas del juego en el nivel de dificultad 0 para comprobar cuál es la distribución de clusters que mejor se adapta al problema. Los resultados de este se exponen a continuación:

Semilla	**3 Clusters**	**5 Clusters**	**6 Clusters**	**11 Clusters**
0	682	2506	1122	1149
1	415	4161	2453	2561
2	562	6382	2629	1370
3	1393	2357	1086	1780
4	297	3695	1074	951
5	304	481	348	316
6	280	4676	2566	2375
7	423	5013	2631	1482
8	367	1185	1368	2511
9	565	354	274	287
TOTAL	**5288**	**30810**	**15551**	**14782**
MEDIA	**528,8**	**3081**	**1555,1**	**1478,2**

Ilustración 35: Puntuaciones obtenidas por las distribuciones de Clusters

A continuación se expone un gráfico para ver estos resultados de forma más visual:

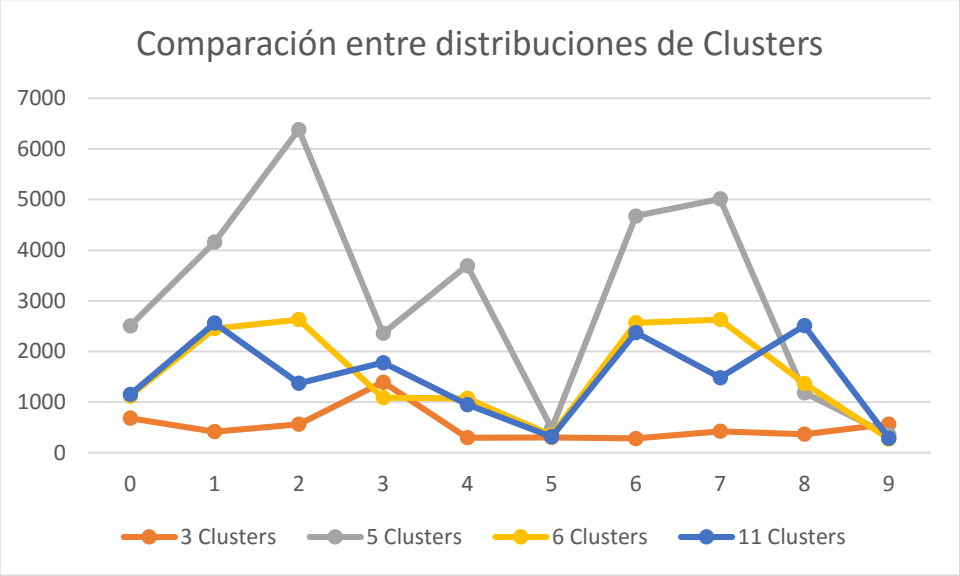

Ilustración 36: Comparación entre distribuciones de Clusters

Como se puede observar de forma clara tanto en la suma de puntuaciones como en el gráfico, la distribución que logra una mayor puntuación, y por tanto se adapta mejor, es la de 5 Clusters, cuya tabla Q se usará en el agente Q-Learning para la evaluación de este. A su vez, los valores de los parámetros del algoritmo y de ciclos que se han usado para obtenerla han sido los siguientes:

- **Número de ciclos de entrenamiento**: 3
- **Tasa de aprendizaje (α)**: 0.225
- **Tasa de descuento (γ)**: 0.9

6.1.4.3 Evolución del aprendizaje

A su vez, para verificar que el aprendizaje offline se ha realizado de forma correcta, se ha evaluado el avance del mismo durante la fase de entrenamiento observando la evolución en la puntuación del agente. Esto se ha llevado a cabo leyendo las tuplas por conjuntos, es decir, sobre el conjunto total primero se leen unas pocas y se ejecuta el agente obteniendo un resultado, a continuación se añaden nuevas tuplas a las ya leídas y se vuelve a realizar la ejecución, y así sucesivamente hasta haberlas leído todas. Si el aprendizaje se realiza correctamente, la lectura de un nuevo conjunto de tuplas debería suponer una mejoría en la puntuación obtenida por el agente. Los resultados que se muestran a continuación pertenecen a la puntuación obtenida en una única ejecución por cada agente del nivel de dificultad 0 y semilla 1. Estos se han conseguido dividiendo en 3 el conjunto total de tuplas y manteniendo los mismos parámetros de entrenamiento de cada algoritmo, de esta manera se distinguen aquellas que tuplas se leen al comenzar el aprendizaje, las que se leen a mitad de

proceso y las que se leen en último lugar, lo que permite observar las diferencias entre ellas y la influencia que tiene cada sector en el entrenamiento:

- Q-Learning:

Conjunto	**Puntuación (Weighted Fitness)**
Conjunto 1	424
Conjunto 1 + Conjunto 2	3815
Conjunto total	4161

Tabla 35: Resultados de la evolución del aprendizaje (Q-Learning)

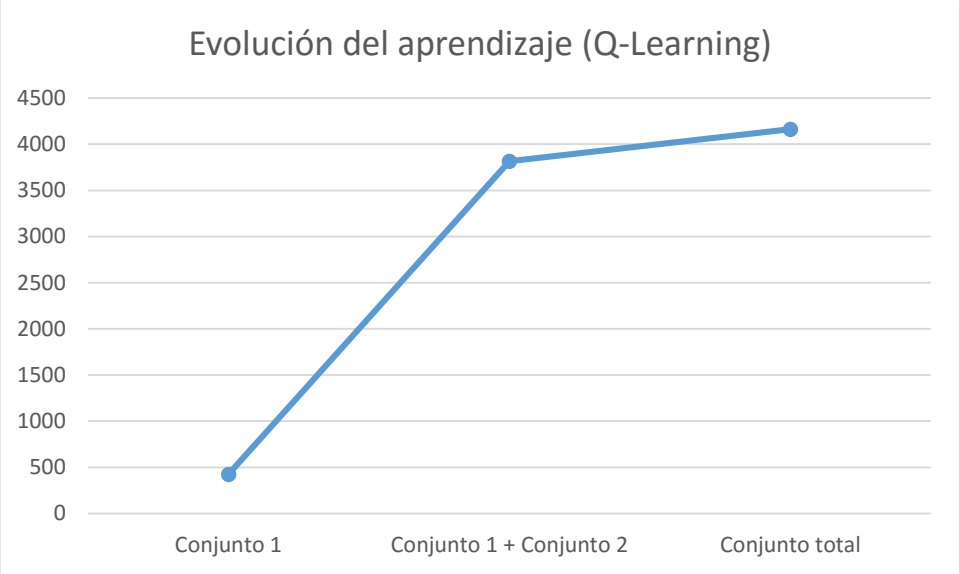

Ilustración 37: Curva de aprendizaje Q-Learning

- Q-Learning Aproximado:

Conjunto	**Puntuación (Weighted Fitness)**
Conjunto 1	424
Conjunto 1 + Conjunto 2	424
Conjunto total	4289

Tabla 36: Resultados de la evolución del aprendizaje (Q-Learning Aproximado)

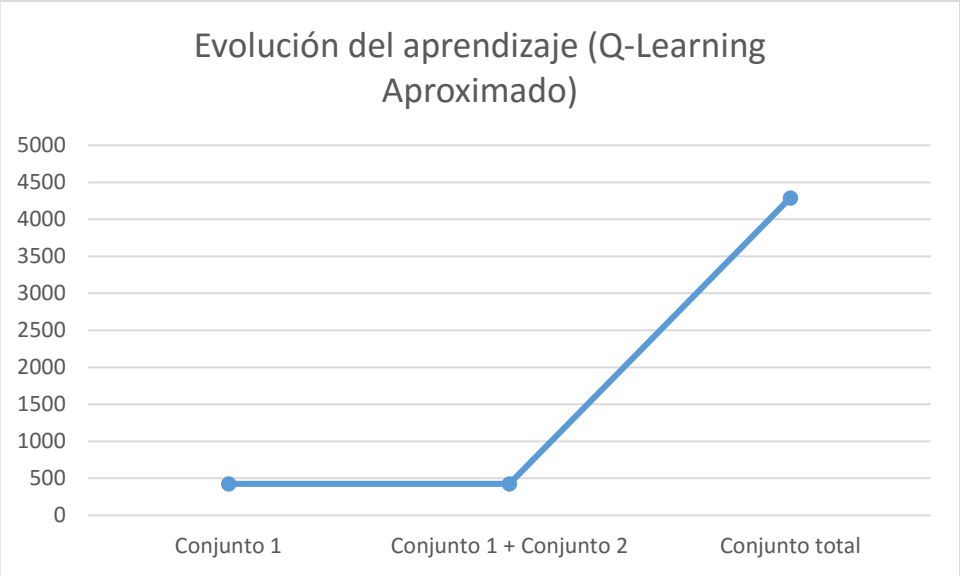

Ilustración 38: Curva de aprendizaje Q-Learning Aproximado

Como se puede apreciar tanto en Q-Learning como en Q-Learning Aproximado, los resultados se corresponden con lo esperado: la lectura incremental de tuplas supone un aumento del refuerzo, por lo tanto, el aprendizaje se realiza de forma correcta. Cabe comentar que, pese a lo dicho anteriormente, hay ciertas diferencias en el aprendizaje de ambos algoritmos: en el caso de Q-Learning la curva de aprendizaje es más progresiva y, sin embargo, en la versión aproximada se produce un escalón con la lectura del conjunto final. La explicación a esto reside en que en el último sector de tuplas se encuentran aquellas que ejecutan las acciones retroceder y saltar hacia atrás, las cuales tienen mucha influencia en el agente Q-Learning Aproximado mientras que el agente Q-Learning no consigue aprender apenas nada de ellas tal y como se podrá observar en el apartado Resultados y análisis.

6.2 Resultados y análisis

A continuación se muestran los resultados obtenidos de las pruebas y criterios establecidos anteriormente:

Semilla	QAgentTFG	BasicAAAgent	TFGAgent
0	2506	6931	8242
1	4161	6417	4289
2	6382	1860	4227
3	2357	8040	1376
4	3695	3962	3786
5	481	7846	4307
6	4676	7936	5072
7	5013	1769	4511
8	1185	1256	8540
9	354	300	300
TOTAL	**30810**	**46317**	**44650**
MEDIA	**3081**	**4631,7**	**4465**

Tabla 37: Puntuaciones obtenidas por los agentes

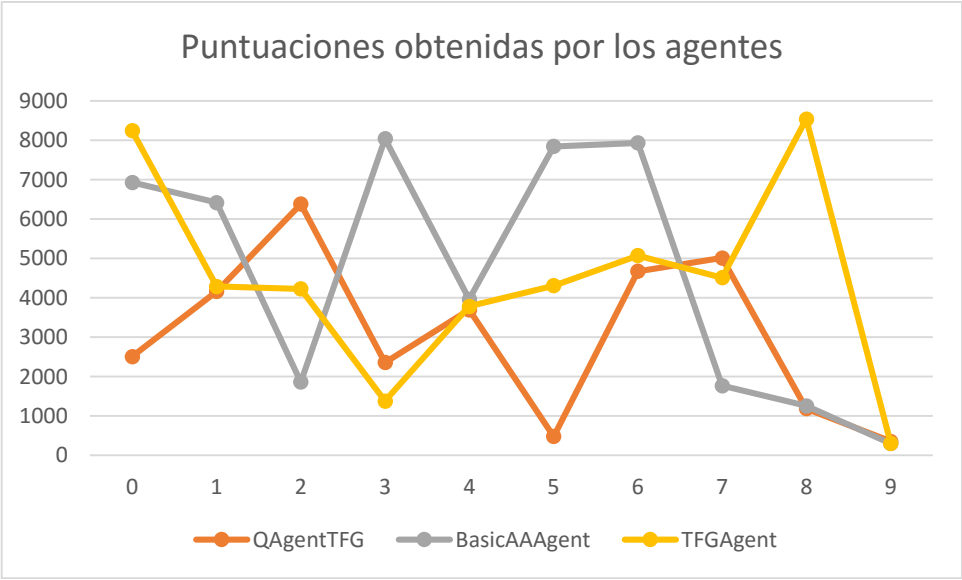

Ilustración 39: Gráfico de puntuaciones obtenidas

Adicionalmente se muestra un desglose de las puntuaciones obtenidas que ayudarán a analizar mejor lo sucedido en cada nivel:

Semilla 0	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	18148 ms	34791 ms	49403 ms
Weighted Fitness	2506	6931	8242
Mario Status	Loss…(atascado)	Loss...	WIN!
Mario Mode	small	small	small
Collisions with creatures	2	3	2
Passed (Cells, Phys)	99 of 256, 1596 of 4096 (38% passed)	240 of 256, 3849 of 4096 (93% passed)	256 of 256, 4096 of 4096 (100% passed)
Time Spent(marioseconds)	200	56	80
Time Left(marioseconds)	0	143	119
Coins Gained	40 of 285 (14% collected)	84 of 285 (29% collected)	85 of 285 (29% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 2 found (0% collected)	0 of 0 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 1 found (0% collected)	0 of 1 found (0% collected)
kills Total	5 of 40 found (12%)	11 of 40 found (27%)	15 of 40 found (37%)
kills By Fire	kills By Fire : 0	0	0
kills By Shell	kills By Shell : 0	0	0
kills By Stomp	kills By Stomp : 5	11	15

Ilustración 40: Puntuación obtenida en la semilla 0

Semilla 1	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	16317 ms	33597 ms	39360 ms
Weighted Fitness	4161	6417	4289
Mario Status	Loss...	Loss...	Loss…(atascado/bug)
Mario Mode	small	small	small
Collisions with creatures	3	3	2
Passed (Cells, Phys)	102 of 256, 1635 of 4096 (39% passed)	202 of 256, 3243 of 4096 (78% passed)	159 of 256, 2555 of 4096 (62% passed)
Time Spent(marioseconds)	26	54	200
Time Left(marioseconds)	173	145	0
Coins Gained	41 of 304 (13% collected)	82 of 304 (26% collected)	51 of 304 (16% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 1 found (0% collected)	0 of 2 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 0 found (0% collected)
kills Total	9 of 46 found (19%)	13 of 46 found (28%)	17 of 46 found (36%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	9	13	17

Ilustración 41: Puntuación obtenida en la semilla 1

Semilla 2	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	28904 ms	10326 ms	18735 ms
Weighted Fitness	6382	1860	4227
Mario Status	Loss...	Loss...(atascado)	Loss...
Mario Mode	small	FIRE	small
Collisions with creatures	3	0	3
Passed (Cells, Phys)	200 of 256, 3204 of 4096 (78% passed)	61 of 256, 987 of 4096 (23% passed)	95 of 256, 1523 of 4096 (37% passed)
Time Spent(marioseconds)	47	49	30
Time Left(marioseconds)	153	150	169
Coins Gained	58 of 245 (23% collected)	37 of 245 (15% collected)	44 of 245 (17% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 0 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 1 found (0% collected)	0 of 1 found (0% collected)
kills Total	19 of 51 found (37%)	4 of 51 found (7%)	12 of 51 found (23%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	19	4	12

Ilustración 43: Puntuación obtenida en la semilla 2

Semilla 3	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	6436 ms	34294 ms	34833 ms
Weighted Fitness	2357	8040	1376
Mario Status	Loss...	WIN!	Loss...(atascado/bug)
Mario Mode	small	FIRE	FIRE
Collisions with creatures	3	0	0
Passed (Cells, Phys)	49 of 256, 791 of 4096 (19% passed)	256 of 256, 4096 of 4096 (100% passed)	48 of 256, 772 of 4096 (18% passed)
Time Spent(marioseconds)	10	55	200
Time Left(marioseconds)	189	144	0
Coins Gained	0 of 188 (0% collected)	66 of 188 (35% collected)	0 of 188 (0% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 0 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 1 found (0% collected)	0 of 0 found (0% collected)
kills Total	1 of 52 found (1%)	12 of 52 found (23%)	10 of 52 found (19%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	1	12	10

Ilustración 42: Puntuación obtenida en la semilla 3

Semilla 4	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	14881 ms	19209 ms	60592 ms
Weighted Fitness	3695	3962	3786
Mario Status	Loss...	Loss...	Loss...(atascado/bug)
Mario Mode	small	small	small
Collisions with creatures	3	3	2
Passed (Cells, Phys)	99 of 256, 1587 of 4096 (38% passed)	122 of 256, 1962 of 4096 (47% passed)	150 of 256, 2400 of 4096 (58% passed)
Time Spent(marioseconds)	24	31	200
Time Left(marioseconds)	175	168	0
Coins Gained	24 of 326 (7% collected)	41 of 326 (12% collected)	63 of 326 (19% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 1 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 1 found (0% collected)	0 of 0 found (0% collected)
kills Total	6 of 41 found (14%)	0 of 41 found (0%)	7 of 41 found (17%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	6	0	7

Ilustración 44: Puntuación obtenida en la semilla 4

Semilla 5	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	10047 ms	41515 ms	22220 ms
Weighted Fitness	481	7846	4307
Mario Status	Loss...(atascado)	WIN!	Loss...
Mario Mode	FIRE	Large	small
Collisions with creatures	0	1	3
Passed (Cells, Phys)	17 of 256, 283 of 4096 (6% passed)	256 of 256, 4096 of 4096 (100% passed)	116 of 256, 1867 of 4096 (45% passed)
Time Spent(marioseconds)	200	67	36
Time Left(marioseconds)	0	132	163
Coins Gained	5 of 262 (1% collected)	72 of 262 (27% collected)	40 of 262 (15% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 3 found (0% collected)
Flowers Devoured	0 of 1 found (0% collected)	0 of 3 found (0% collected)	1 of 1 found (100% collected)
kills Total	1 of 36 found (2%)	9 of 36 found (25%)	8 of 36 found (22%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	1	9	8

Ilustración 45: Puntuación obtenida en la semilla 5

Semilla 6	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	19721 ms	38311 ms	51538 ms
Weighted Fitness	4676	7936	5072
Mario Status	Loss...	WIN!	Loss...(atascado/bug)
Mario Mode	small	small	FIRE
Collisions with creatures	3	2	0
Passed (Cells, Phys)	133 of 256, 2132 of 4096 (51% passed)	256 of 256, 4096 of 4096 (100% passed)	209 of 256, 3348 of 4096 (81% passed)
Time Spent(marioseconds)	32	62	200
Time Left(marioseconds)	167	137	0
Coins Gained	35 of 227 (15% collected)	67 of 227 (29% collected)	43 of 227 (18% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 0 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 1 found (0% collected)
kills Total	12 of 46 found (26%)	12 of 46 found (26%)	18 of 46 found (39%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	12	12	18

Ilustración 46: Puntuación obtenida en la semilla 6

Semilla 7	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	20170 ms	12077 ms	55432 ms
Weighted Fitness	5013	1769	4511
Mario Status	Loss...	Loss...(atascado)	Loss...(atascado/bug)
Mario Mode	small	FIRE	small
Collisions with creatures	3	0	2
Passed (Cells, Phys)	137 of 256, 2197 of 4096 (53% passed)	71 of 256, 1147 of 4096 (27% passed)	179 of 256, 2865 of 4096 (69% passed)
Time Spent(marioseconds)	32	200	200
Time Left(marioseconds)	167	0	0
Coins Gained	52 of 253 (20% collected)	18 of 253 (7% collected)	59 of 253 (23% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 2 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 3 found (0% collected)
kills Total	12 of 39 found (30%)	5 of 39 found (12%)	13 of 39 found (33%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	12	5	13

Ilustración 47: Puntuación obtenida en la semilla 7

Semilla 8	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	10097 ms	9844 ms	45714 ms
Weighted Fitness	1185	1256	8540
Mario Status	Loss...(atascado)	Loss...(atascado)	WIN!
Mario Mode	small	small	Large
Collisions with creatures	2	2	1
Passed (Cells, Phys)	47 of 256, 763 of 4096 (18% passed)	47 of 256, 764 of 4096 (18% passed)	256 of 256, 4096 of 4096 (100% passed)
Time Spent(marioseconds)	200	200	74
Time Left(marioseconds)	0	0	125
Coins Gained	23 of 390 (5% collected)	24 of 390 (6% collected)	102 of 390 (26% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 0 found (0% collected)	0 of 0 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 1 found (0% collected)	0 of 3 found (0% collected)
kills Total	1 of 36 found (2%)	2 of 36 found (5%)	14 of 36 found (38%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	1	2	14

Ilustración 48: Puntuación obtenida en la semilla 8

Semilla 9	QAgentTFG	BasicAAAgent	TFGAgent
Evaluation lasted	4324 ms	5213 ms	5385 ms
Weighted Fitness	354	300	300
Mario Status	Loss...(atascado)	Loss...(atascado)	Loss...
Mario Mode	FIRE	FIRE	FIRE
Collisions with creatures	0	0	0
Passed (Cells, Phys)	13 of 256, 220 of 4096 (5% passed)	13 of 256, 219 of 4096 (5% passed)	13 of 256, 220 of 4096 (5% passed)
Time Spent(marioseconds)	200	84	200
Time Left(marioseconds)	0	115	0
Coins Gained	1 of 191 (0% collected)	1 of 191 (0% collected)	1 of 191 (0% collected)
Hidden Blocks Found	0 of 0 (0% found)	0 of 0 (0% found)	0 of 0 (0% found)
Mushrooms Devoured	0 of 0 found (0% collected)	0 of 1 found (0% collected)	0 of 0 found (0% collected)
Flowers Devoured	0 of 0 found (0% collected)	0 of 2 found (0% collected)	0 of 0 found (0% collected)
kills Total	1 of 41 found (2%)	0 of 41 found (0%)	0 of 41 found (0%)
kills By Fire	0	0	0
kills By Shell	0	0	0
kills By Stomp	1	0	0

Ilustración 49: Puntuación obtenida en la semilla 9

Como se puede observar en los resultados, a primera vista y basándose solamente en la puntuación, el mejor agente parece ser BasicAAAgent (el agente preprogramado) seguido muy de cerca por TFGAgent (Q-Learning Aproximado), pero sin embargo esta observación no es del todo exacta ni completa, por lo que a continuación se va a comentar el comportamiento que ha tenido cada agente a lo largo de los distintos niveles evaluados:

- **QAgentTFG**: Como se puede observar a partir de los resultados este agente consigue un avance y número de muertes relativamente bueno pero no termina de brillar en cuanto a puntuación, a pesar de superar en alguna ocasión a los otros agentes, dado que el único pico que alcanza en la prueba no es demasiado alto, como se puede observar en el gráfico. Esto se debe a que se atasca en bastantes ocasiones y no ha aprendido ninguna acción de retroceder probablemente porque el algoritmo ha terminado de ajustar bien el modelo del problema contenido en las tuplas de experiencia.

- **BasicAAAgent**: Como se puede apreciar observando los resultados obtenidos al final de cada semilla la mayor parte de la puntuación obtenida por este agente proviene del avance, sin apenas centrarse en matar enemigos o coger setas, flores o monedas (las monedas que recolecta son las que encuentra a su paso). A su vez, al tener únicamente programadas las acciones de avanzar y saltar hacia delante tiene muy pocas ocasiones de quedarse atascado, lo que ocasiona que llegue a pasarse los niveles más veces que sus competidores, ganando en estos casos el bonus por partida ganada (+1024 puntos) e inflando mucho su puntuación. Sin embargo cuando este queda en un nivel a la par en avance que los otros agentes, o incluso un poco por encima, se puede observar que los agentes con inteligencia artificial le superan con creces en el número de muertes logradas en proporción, y de igual manera, cuando TFGAgent logra ganar la partida supera en puntuación a todas las ganadas por BasicAAAgent como se puede observar de forma clara en el gráfico.

- **TFGAgent**: A partir de los resultados expuestos se puede ver que este agente se caracteriza por tener un buen avance, una cantidad de monedas recogidas bastante buena, y sobre todo por conseguir un número de muertes bastante elevado en proporción a los otros agentes. En cuanto a puntuación es un agente muy bueno, destaca por encima del agente que aplica Q-Learning en casi todos los niveles y consigue los dos picos más altos de puntuación de toda la evaluación. Sin embargo, su principal defecto es que en bastantes ocasiones se queda atascado o bug, ya sea porque hay un enemigo debajo del terreno (bug del simulador), o porque en algunos casos cambia de acción demasiado deprisa y no llega nunca a matar enemigos o coger elementos que están en otro nivel de la escena. Este último caso podría solucionarse retrasando la frecuencia con la que se reciben estados y se ejecutan acciones, pero esto provocaría que el agente dejase de tener en cuenta ciertos elementos del escenario y que no reaccionase bien ante estos cuando en realidad tiene capacidad. Por ejemplo, en el caso de la semilla 5 este agente logra coger una flor de fuego, si retrasamos a una frecuencia de 2 ticks la emisión de acciones/recepción de estados, el agente la ignoraría y jamás la

cogería. Cabe comentar también que durante estas pruebas, cuando se ha producido el bug, se ha tenido que forzar el final de la ejecución mediante un "time out", es decir, dejar el tiempo restante para acabar el nivel a 0, lo que provoca que en estos casos no se sume a Weighted Fitness el bono por tiempo restante y la puntuación sea ligeramente más baja que la que debería obtener realmente.

A modo de conclusión, podría considerarse que el agente BasicAAAgent es el mejor en términos de avance, pero el agente más equilibrado y que más potencial tiene en cuanto a la ganancia de puntuación, que es el aspecto que persigue este proyecto, es TFGAgent. En cuanto a la comparación entre Q-Learning Aproximado y Q-Learning, se puede observar que ambos problemas adaptan el problema correctamente, pero sin embargo, es más precisa la adaptación conseguida por la versión aproximada, dado ha conseguido aprender a realizar más acciones, como retroceder, y a llevarlas a cabo mediante prioridades basadas en la distancia a los elementos. De esta manera, durante la ejecución de TFGAgent se puede ver que Mario retrocede a matar a aquellos enemigos que haya dejado atrás, pero que sin embargo no lo hace por sistema: si el enemigo que hay detrás está muy alejado o hay monedas más adelante, el agente decide ignorarlo porque no le reporta tanto beneficio como continuar hacia el final del nivel. Por otra parte, estas dos versiones presentan un modelo mucho más sólido frente a las distintas situaciones a las que se pueda enfrentar Mario durante los distintos niveles que el preprogramado, dado que este último solo reacciona ante aquellas sentencias que tenga programadas mientras que los agentes que utilizan Inteligencia Artificial son capaces de resolver situaciones más complejas e incluso imprevistas, siempre que estén comprendidas en su espacio de estados.

7 Planificación y presupuesto

7.1 Planificación

La planificación propuesta para la realización del proyecto se basa en el Modelo en Cascada descrito en el apartado 5.5.1 Metodología empleada de este documento. En la siguiente tabla, se incluyen y describen las distintas fases y tareas proyectadas:

PLANIFICACIÓN	
Fase 1: Estudio	
Tarea 1.1	Repaso de Aprendizaje Automático
Tarea 1.2	Repaso del simulador Mario AI y software Q-Learning
Tarea 1.3	Estudio de Q-Learning Aproximado
Fase 2: Análisis	
Tarea 2.1	Extracción de requisitos
Tarea 2.2	Realización de los casos de uso
Tarea 2.3	Planificación y presupuesto
Tarea 2.4	Documentación
Fase 3: Diseño	
Tarea 3.1	Diseño de la arquitectura del sistema
Tarea 3.2	Adaptación de los algoritmos a los sistemas MarioAI y Q-Learning
Tarea 3.3	Primera iteración sobre la representación del problema
Tarea 3.4	Segunda iteración sobre la representación del problema
Fase 4: Implementación	
Tarea 4.1	Implementación de los algoritmos en el software Q-Learning
Tarea 4.2	Implementación de los agentes de entrenamiento
Tarea 4.3	Implementación del agente automático
Tarea 4.4	Conectividad de los agentes con los sistemas Q-Learning y MarioAI
Fase 5: Pruebas y evaluación	
Tarea 5.1	Diseño de las pruebas
Tarea5.2	Realización de las pruebas
Tarea 5.3	Evaluación
Fase 6: Revisión	
Tarea 6.1	Revisión del documento

Tabla 38: Fases y tareas de la planificación

A continuación, se adjunta un diagrama de Gantt que permite observar de forma gráfica la duración asignada a cada una de las tareas expuestas en la tabla anterior:

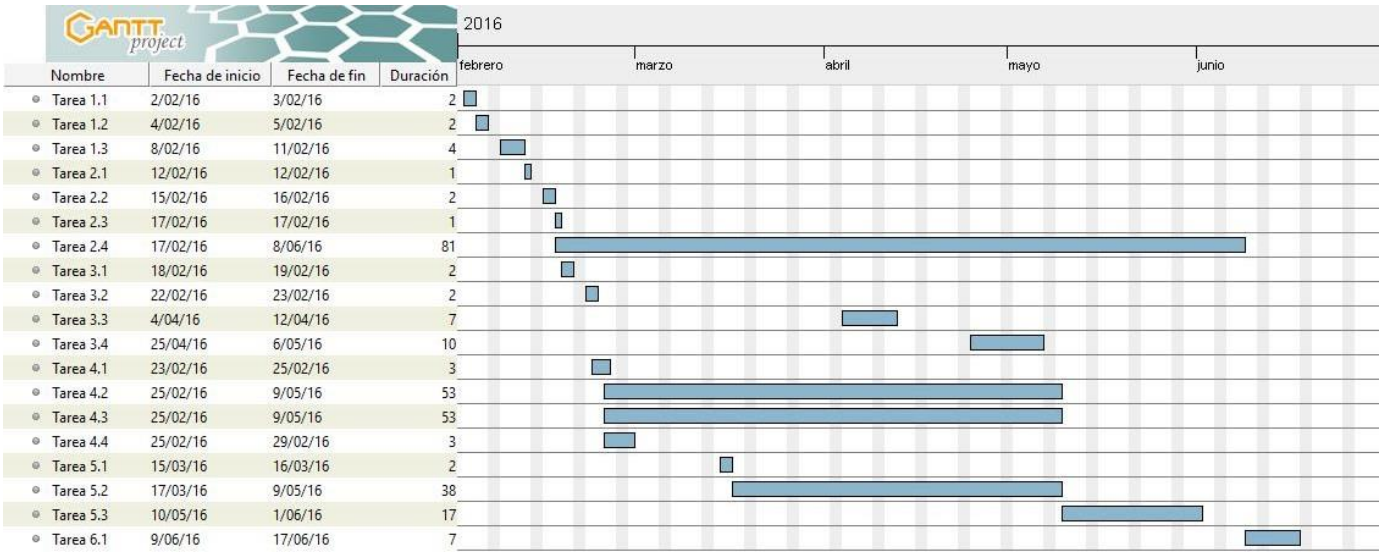

Ilustración 50: Diagrama de Gantt con la planificación del proyecto

7.2 Presupuesto

En este apartado se detalla el coste total del proyecto incluyendo el equipo de trabajo, licencias software y salario sobre la duración del proyecto, la cual finalmente ha coincidido la planificación diseñada. Como la mayoría del proyecto se ha desarrollado utilizando programas con licencia Open Source, buena parte de estos no se contabilizan a continuación al no tener coste alguno.

7.2.1 Gastos de personal.

El personal consta de un Ingeniero Informático que asume los roles que se detallan en la siguiente tabla y a los cuales corresponden los siguientes costes, teniendo en cuenta una jornada de 5 horas al día durante los cuatro meses que dura el proyecto:

Rol	*Coste/hora (€)*	*Días dedicados**	*Total horas dedicadas*	*Coste total (€)*
Analista/Diseñador	33	97	485	16005
Programador	25	81	405	10125
Responsable de Documentación	15	88	440	6600
TOTAL				**32730€**

Tabla 39: Gastos de personal

** Días dedicados calculados a partir de las fases establecidas en el diagrama de Gantt. Se ha asignado a cada uno de los roles las siguientes fases: Analista/Diseñador (Estudio, Análisis, Diseño), Programador (Estudio, Implementación, Pruebas), Responsable de documentación (Análisis (sólo tarea de documentación), Revisión).*

7.2.2 Amortización del Hardware y el Software.

A continuación se detalla la amortización del hardware y del software sin licencia Open Source de acuerdo a la duración del proyecto:

Concepto	*Coste (€)*	*% Uso dedicado al proyecto*[*1]	*Dedicación (meses)*	*Periodo de depreciación (meses)*	*Coste imputable (€)*[*2]
PC Sobremesa	849	100	4	60	56,6
Windows 10 Home	99,50	100	4	60	6,63
Microsoft Office 365	29,95	100	4	12	9,98
TOTAL					**73,21€**

Tabla 40: Gastos de Hardware y Software

*[*1] Porcentaje de uso en el proyecto del software o hardware especificado. En el caso de Microsoft Office 365 se le aplica un 100% de uso al estar constantemente documentando toda durante todas las fases del proyecto.*

*[*2] Coste imputable calculado a partir de la fórmula de la amortización:*

$$\frac{Dedicación * Coste * \%Uso}{Periodo\ de\ depreciación}$$

7.2.3 Costes Indirectos

Los costes indirectos hacen referencia a gastos asociados al desarrollo del proyecto tales como el material de oficina, mobiliario, mantenimiento de los equipos, luz y agua. Se estima que esta cantidad supone aproximadamente un **15% sobre la suma del gasto del personal más la amortización del equipo (sin incluir I.V.A)**.

7.2.4 Coste total

Finalmente, el coste total que supone el proyecto, teniendo en cuenta todos los factores detallados con anterioridad, es el siguiente:

Concepto	*Coste (€)*
Gasto del Personal	32730
Amortización equipo	73,21
Costes Indirectos	4920,48
Subtotal	37723,69
I.V.A (21%)	7921,97
TOTAL	**45645,66**

Tabla 41: Coste total del proyecto

8 Conclusiones y líneas futuras de trabajo

Como se ha podido comprobar a lo largo de este documento, se han cumplido todos los objetivos propuestos para este proyecto:

- ✓ Se ha desarrollado un agente basado en Q-Learning Aproximado con un buen funcionamiento capaz de superar niveles obteniendo una puntuación alta y, a su vez, ha quedado comprobado que este algoritmo es capaz de aprender de forma satisfactoria el modelo del problema, siendo este agente el que mejor ha aprendido todas las acciones posibles, tal y como ha quedado patente en la sección de pruebas de este documento.

- ✓ Se ha trabajado la técnica de Q-Learning y Clustering realizando un estudio teórico de las mismas y aplicándolas con éxito sobre un nuevo agente, reduciendo el espacio de estados tan grande que planteaba el objetivo de puntuación del problema.

- ✓ Se ha razonado y demostrado cómo la técnica de Q-Learning Aproximado puede ser una alternativa sólida al Clustering de cara a trabajar con estados muy grandes, lo que permite mucha más precisión a la hora de representar ciertas situaciones en los problemas.

- ✓ Se ha realizado una comparación a partir de los resultados obtenidos por las distintas técnicas empleadas y se ha evaluado cómo de bien se han adaptado al problema, comentando además las ventajas obtenidas por cada una.

- ✓ Tal y como se ha puesto en este documento, se ha llevado a cabo una experimentación en la generación y tratamiento de los datos del problema, tratando de explorar todas las vías posibles para su resolución. De esta manera se ha probado, por ejemplo, el uso de características binarias para los estados, distancia euclídea o diferentes tamaños de matrices.

Cabe comentar que, aunque el resultado final conseguido por los agentes desarrollados ha sido acorde a las expectativas, debido a que la experimentación del problema fue muy larga, no ha dado tiempo a incluir algunas mejoras que probablemente ayudarían a aliviar algunas carencias que estos padecen. Dichas mejoras se podrían incluir como líneas futuras de trabajo y son las siguientes:

- **Inclusión del aprendizaje on-line**: Sería interesante que los agentes pudiesen actualizar su tabla de pesos o Tabla Q conforme avanza la partida, dado que esto podría resolver muchos de los atascos que sufren los agentes al penalizar las acciones llevadas a cabo en esos casos y a su vez potenciaría aquellas acciones que reportan más beneficios.

- **Agente E-Greedy para tomar ejemplos**: Relacionado con el punto anterior, este agente se comenzó a desarrollar para incluir el aprendizaje online, pero tuvo que dejarse de lado finalmente para poder afrontar toda la experimentación que ha requerido el desarrollo de este proyecto. La idea que perseguía este agente era la obtención de ejemplos explorando el entorno de forma aleatoria al principio para poco a poco construir, mediante un algoritmo como Q-Learning, una tabla de decisiones de la que tomar las siguientes acciones de exploración. De esta manera, los ejemplos que se obtengan podrían representar situaciones que no se dan en otros de los agentes de este proyecto empleados para el entrenamiento, por lo que probablemente otorgarían más capacidades al agente final dotándole de más inteligencia e incluso haciéndole capaz de rivalizar con un humano.

- **Uso de características basadas en coordenadas:** Se propuso un modelo de características que finalmente no se llegó a aplicar por cuestiones de tiempo, que sustituyese a aquellas que localizan basándose en distancias ciertos elementos de la escena, como por ejemplo enemigoDetras o enemigoDelante, por unas que expresasen las coordenadas donde se hallan tales elementos. De esta manera el anterior ejemplo quedaría como enemigoX y enemigoY, lo cual, aplicado las demás características similares, reduciría el espacio de estados y eliminaría a su vez cierta redundancia existente ellas. Otra ventaja que tiene este modelo es que, al tratarse de coordenadas cartesianas, estaría directamente normalizado lo que supondría ahorrarse el proceso de normalización.

9 Referencias

[1] S. Arango, «YoungMarketing.co - Historia de Super Mario Bros,» 7 Noviembre 2013. [En línea]. Available: http://www.youngmarketing.co/la-historia-de-super-mario-bros/. [Último acceso: 11 4 2016].

[2] Super Mario Wiki, «Super Mario Wiki - Super Mario Bros,» 15 Noviembre 2015. [En línea]. Available: http://es.mario.wikia.com/wiki/Super_Mario_Bros.. [Último acceso: 2016 Abril 11].

[3] S. Karakovskiy y J. Togelius, «JulianTogelius.com,» 2009. [En línea]. Available: http://julian.togelius.com/mariocompetition2009/CIG2009Competition.pdf. [Último acceso: 26 Marzo 2016].

[4] S. Karakovskiy y J. Togelius, «The MarioAI Bechmark and Competitions; JulianTogelius.com,» 2012. [En línea]. Available: http://julian.togelius.com/Karakovskiy2012The.pdf. [Último acceso: 14 Mayo 2016].

[5] F. Sánchez Caparrini, «Introducción al Aprendizaje Automático,» 15 Julio 2015. [En línea]. Available: http://www.cs.us.es/~fsancho/?e=75. [Último acceso: 2 Marzo 2016].

[6] F. Fernández Rebollo y D. Borrajo Millán, «Open Course Ware: Aprendizaje por Refuerzo,» 27 Febrero 2009. [En línea]. Available: http://ocw.uc3m.es/ingenieria-informatica/aprendizaje-automatico/material-de-clase-1/aa-ocw-refuerzo.pdf/view. [Último acceso: 2016 Marzo 19].

[7] F. Fernández Rebollo y D. Borrajo Millán, «Open Course Ware: Procesos de Decisión de Markov,» 27 Febrero 2009. [En línea]. Available: http://ocw.uc3m.es/ingenieria-informatica/aprendizaje-automatico/material-de-clase-1/aa-ocw-mdp.pdf/view. [Último acceso: 19 Marzo 2016].

[8] R. Hermoso y M. Vasirani, «Aprendizaje por refuerzo,» Universidad Rey Juan Carlos, 2011. [En línea]. Available: http://www.ia.urjc.es/cms/sites/default/files/userfiles/file/ia4/2011/IA4_[Refuerzo]%281%29.pdf. [Último acceso: 16 5 2016].

[9] D. Klein, «University of California, Berkeley,» 10 5 2010. [En línea]. Available: https://inst.eecs.berkeley.edu/~cs188/fa10/slides/FA10%20cs188%20lecture%2012%20--%20reinforcement%20learning%20II%20%286PP%29.pdf. [Último acceso: 2 Marzo 2016].

[10] colaboradores de Wikipedia, «Algoritmo de agrupamiento,» 6 Noviembre 2015. [En línea]. Available: https://es.wikipedia.org/wiki/Algoritmo_de_agrupamiento. [Último acceso: 16 Mayo 2016].

[11] Wikipedia contributors, «Cluster analysis,» 14 5 2016. [En línea]. Available: https://en.wikipedia.org/wiki/Cluster_analysis. [Último acceso: 2016 5 16].

[12] The University of Waikato, «Weka 3: Data Mining Software in Java,» The University of Waikato, [En línea]. Available: http://www.cs.waikato.ac.nz/ml/weka/index.html. [Último acceso: 20 5 2016].

[13] University of California, Berkeley, «Intro to AI -- Course Materials. Project 3: Reinforcement Learning,» Berkeley, 26 8 2014. [En línea]. Available: http://ai.berkeley.edu/reinforcement.html#Q8. [Último acceso: 15 2 2016].

[14] Open Source Iniciative, «Artistic License 2.0,» 2006. [En línea]. Available: https://opensource.org/licenses/Artistic-2.0. [Último acceso: 24 3 2016].

[15] colaboradores de Wikipedia, «Distancia euclidiana,» 26 Enero 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Distancia_euclidiana. [Último acceso: 18 Mayo 2016].

[16] C. Ble, «Diseño ágil con TDD,» 2013.

[17] D. Cantone, «Ciclo de vida del Software,» de *Implementación y Debugging*, Buenos Aires, MP Ediciones, 2008, p. 3220.

[18] G. Génova Fuster, «Ingeniería del Software: Un enfoque crítico,» Universidad Carlos III de Madrid, Leganés, 2012.